Simplification de contraintes d'integrite relationnelles par generation de pre-tests differentiels

Eric Simon

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 742

SIMPLIFICATION DE CONTRAINTES D'INTEGRITE RELATIONNELLES PAR GENERATION DE PRE-TESTS DIFFERENTIELS

Eric SIMON

NOVEMBRE 1987

SIMPLIFICATION DE CONTRAINTES D'INTEGRITE RELATIONNELLES PAR GENERATION DE PRE-TESTS DIFFERENTIELS

Eric Simon

INRIA, Rocquencourt, BP. 105
78153 Le Chesnay cedex

Résumé :

Ce papier présente une méthode originale de simplification de contraintes d'intégrité basée sur la notion de pré-tests différentiels. Cette méthode possède plusieurs propriétés : (i) la simplification des contraintes est effectuée une fois pour toutes au moment de leur définition, (ii) la méthode permet un contrôle préventif des incohérences dans la base de données, (iii) la méthode est générale : elle supporte une large classe d'assertions multi-relations, multi-variables et accepte des mises à jour ensembliste d'une relation, et (iv) une typologie des assertions permet de déterminer une sous classe d'assertions pour laquelle des pré-tests différentiels simplifiés sont obtenus; cette sous classe recouvre la famille des dépendances. Nous présentons également des algorithmes généraux de mise en oeuvre de la méthode ainsi que les algorithmes qui ont été implantés dans le système SABRINA.

Mots - clés : calcul relationnel, contrainte d'intégrité, assertion, pré-test différentiel, simplification de contrainte, algorithmes.

SIMPLIFICATION OF RELATIONAL INTEGRITY CONSTRAINTS BY GENERATING DIFFERENTIAL PRE-TESTS

Eric Simon

INRIA - Rocquencourt, BP 105
78153 Le Chesnay cedex, France

Abstract :

This paper presents an original method for simplifying integrity constraints in relational database systems. This method is based on differential pre-tests and has several properties : (i) constraint simplification is performed once at the time where assertions are defined, (ii) it allows a prventive control of unconsistencies in the database, (iii) it is general i.e., it supports a large class of multi-relation, multi-variable assertions and accepts set oriented updates of a relation, and (iv) it pemits to isolate a sub-class of assertions for which simplified differential pre-tests are obtained; this sub-class covers the family of database dependencies. We finally present general algorithms for processing the simplification method as well as the algorithms that have been implemented in the SABRINA database system.

Key-words : relational calculus, integrity constraint, assertion, differential pre-test, simplification method, algorithms.

1. INTRODUCTION

Un problème fondamental pour un Système de Gestion de Bases de Données (SGBD) est de garantir la cohérence de la base de données. Une base de données est dite cohérente si elle satisfait un ensemble de contraintes appelées *contraintes d'intégrité sémantique* Maintenir la cohérence d'une base de données nécessite divers mécanismes tels que le contrôle de concurrence, la résistance aux pannes, le contrôle de la protection des données et le contrôle de l'intégrité sémantique des données. Ce dernier assure la cohérence de la base en rejetant les transactions de mise à jour qui conduiraient la base dans un état incohérent, (transactions *invalides*), ou en activant des actions spécifiques sur la base de données qui compensent l'effet de ces transactions. Ainsi, la base de données mise à jour doit satisfaire de nouveau l'ensemble des contraintes d'intégrité. Deux méthodes de base existent pour rejeter les transactions invalides. La première repose sur la *détection* des incohérences. Toute mise à jour, m, est exécutée et l'état de la base de données, D, est changé en D_m. Le contrôle d'intégrité vérifie que toutes les contraintes pertinentes sont satisfaites dans l'état D_m. Si D_m est détecté incohérent, un état cohérent peut être restauré soit en déclenchant l'exécution d'actions compensatrices de m, (on obtient alors un état D_m'), soit en restituant l'état D initial. Cette approche est souvent inefficace(*) parcequ'un gros travail est perdu, (la construction de D_m ou la restauration de D), en cas d'incohérence. La seconde méthode de base repose sur la *prévention* des incohérences. Cette fois, une mise à jour m n'est exécutée que si l'état résultant de la base, D_m, est garanti être cohérent. Une méthode préventive particulière est de déterminer tous les programmes d'application ou toutes les transactions qui conduisent par construction à des états cohérents. Par suite, les seuls programmes autorisés à s'exécuter sont valides. La cohérence est alors assurée par des conditions sur la structure des transactions ou des programmes. Ceci conduit à la notion de *contraintes dynamiques*. Une autre méthode préventive est de définir des conditions portant sur l'état de la base qui résulterait de l'exécution des transactions. La cohérence est ici assurée par des conditions sur les données mises à jour.

En général, les contraintes d'intégrité sémantiques représentent la *connaisance* des propriétés de l'application. Ce sont des règles qui définissent les propriétés statiques ou dynamiques de l'application et qui ne peuvent pas être capturées directement par les concepts d'objet et d'opération du *modèle de données*. Ainsi, le concept de règle de cohérence est-il étroitement relié à celui de modèle de données. Les contraintes d'intégrité se divisent surtout en *contraintes structurelles* et en *contraintes de comportement*. Les premières expriment des propriétés sémantiques de base inhérentes à un modèle de données. Ce sont par exemple des associations (1, n) dans un modèle réseau ou des contraintes d'unicité de clé dans le modèle relationnel. Les contraintes de comportement, quant à elles, régulent le comportement de l'application. Elles sont essentielles dans le processus de conception de la base de données. Elles expriment des associations entre objets telles que les contraintes d'inclusion dans le modèle relationnel ou décrivent les structures et propriétés des

(*) bien qu'aucune étude sérieuse confrontant les deux approches n'ait été publiée.

objets. La variété croissante des applications bases de données et le récent développement d'outils d'aide à la conception puissants [Bouzeghoub86] entraînent le besoin de contraintes d'intégrité complexes qui peuvent enrichir le modèle de données. Deux difficultés majeures ont pourtant gêné le développement de sous systèmes d'intégrité puissants dans les SGBD. Le premier est la définition d'un langage de manipulation de contraintes de haut niveau qui permette à l'utilisateur de formuler simplement sa connaissance de l'application. Le second et principal problème est la conception d'algorithmes suffisamment efficaces pour maintenir la base cohérente en présence de mises à jour.

Le contrôle d'intégrité est apparu avec le traitement des données et a évolué depuis des méthodes procédurales vers des méthodes déclaratives. Les premiers contrôles d'intégrité étaient immergés dans les programmes d'application s'interfaçant avec un sytème de fichiers. Ils ont conduit à des problèmes bien connus tels que la redondance de code, la dépendance programme- données, les faibles performances. Autour des années 70, ces contrôles d'intégrité étaient centralisés dans des procédures bases de données [Codasyl73], stockées comme des programmes systèmes et activées par le SGBD. Plus généralement, les procédures bases de données permettent à l'administrateur de spécifier des règles de protection ou de cohérence. Elles sont implantées dans la plupart des systèmes hiérarchiques ou réseau. Cependant, de telles procédures sont difficiles à définir et à gérer (car peu explicites). Elles sont attachées aux objets internes ce qui les rend statiques et dépendantes physiquement des données. Finalement, une transaction de mise à jour génére l'exécution systématique de toutes les procédures définies dans le schéma externe des objets mis à jour alors que seules certaines d'entre elles ou seuls des morceaux de chacune d'entre elles sont pertinents.

Héritée des langages de programmation, la propriété d'abstraction d'un Type de Donnée Abstrait (TDA), i.e., la caractérisation d'un objet par son type et ses opérations indépendamment de sa représentation physique, a été proposée, par exemple dans [Brodie78], pour exprimer et contrôler les contraintes d'intégrité. Dans [Brodie78], l'expression des contraintes d'intégrité se fait au moyen d'un langage de définition du schéma conceptuel (nommé BETA) indépendamment du modèle de données choisi pour le SGBD. BETA permet la spécification de types de base (entier, réél, ...) et de types complexes obtenus par agrégation et généralisation. Des contraintes de clé, fonctionnelles ou multi-valuées sont exprimées sur les types d'objet par des assertions logiques du premier ordre. De plus, l'axiomatisation des types de données s'exprime également par des assertions. Garantir la cohérence de la base nécessite deux phases distinctes. La première est de *vérifier* que la spécification du schéma est correcte (ex., satisfaisabilité des assertions entre elles, ...). La seconde étape consiste à *valider* un état de la base de données par rapport à la spécification du schéma. La méthode décrite dans [Brodie78] n'adresse que la première étape. Les principaux avantages de la méthode sont l'indépendance logique fournie par les TDA, le caractère extensible des types de données et la possibilité de vérifier la spécification du schéma. Cependant, le contrôle des contraintes référençant plusieurs objets n'est pas optimisé. D'autres approches hybrides basées sur des Réseaux Sémantiques (RS) essaient de tirer avantage des concepts puissants fournis pour codifier la connaissance de l'application [Findler79]. Dans un RS, les objects qui partagent des propriétés

communes sont groupés dans des classes. Ce principe peut être généralisé en définissant récursivement des méta-classes à partir de classes ayant des propriétés communes. Les associations entre objets sont représentées par des arcs prédefinis. Le principal problème reste d'exprimer le comportement de ces associations. la solution proposée dans [Mylopoulos80] utilise un langage (TAXIS) sur un réseau simplifié et associe à chaque noeud des procédures de comportement. Le réseau permet de capturer de nombreuses contraintes (agrégation, généralisation, ...). Les contraintes qui ne peuvent être exprimées par les arcs du réseau sont soit immergées dans les transactions sous la forme de pré ou post conditions soit définies sous forme d'assertions. Les contraintes structurelles attachées aux objets peuvent être optimisées car elles correspondent aux procédures prédéfinies. Cependant, le compilateur TAXIS n'optimise pas toutes les autres contraintes de comportement.

Les méthodes déclaratives sont nées pour répondre à certains des problèmes précédents. L'idée, tout d'abord suggérée dans [Florentin74], est d'exprimer toutes les contraintes d'intégrité comme des assertions du calcul de prédicats dont les variables sont interprétées sur l'état de la base de données. Cette approche permet de facilement déclarer et modifier des contraintes d'intégrité.

Du fait de la complétude de l'algèbre relationnelle par rapport à un sous ensemble du calcul de prédicats du premier ordre (calcul relationnel), il n'est pas surprenant que ces méthodes déclaratives soient étudiées dans les systèmes relationnels. De plus, de nombreux efforts dans la théorie du modèle relationnel ont porté sur la formalisation des propriétés des contraintes structurelles appelées *dépendances pleines* qui correspondent à une large classe d'assertions du calcul relationnel. Le principal problème étudié est celui de *l'implication* i.e., de détecter si une *dépendance* donnée est impliquée par un ensemble d'autres dépendances.

La principale difficulté reste que contrôler la satisfaction des assertions lors d'une mise à jour de la base peut être d'un coût prohibitif. Ceci parce que l'évaluation des assertions nécessite d'accéder à de grands volumes de données qui ne sont pas mis en cause par la mise à jour. De nombreuses solutions ont été explorées en combinant des stratégies d'optimisation. Ces stratégies ont pour but (i) de limiter le nombre d'assertions à vérifier, (ii) de réduire le nombre d'accés aux données pour vérifier une assertion en fonction d'une mise à jour donnée, (iii) de prévenir l'introduction d'incohérences dans la base et (iv) d'effectuer autant que possible les contrôles d'intégrité au moment de la compilation de la mise à jour. Cependant, très peu de solutions ont été implantées. Les méthodes existantes souffrent d'abord d'un manque de généralité soit parcequ'elles sont restreintes à un petit ensemble d'assertions, (des assertions plus complexes auraient un coût prohibitif), soit parce qu'elles ne supportent que des transactions trop restrictives (ex., mise à jour d'un seul tuple dans une relation). Elles sont ensuite peu extensibles à cause de leur étroite dépendance à une implantation particulière dans un environnement sytème.

Dans de précédents papiers, nous avons présenté la conception et l'analyse d'un sous système

d'intégrité [Simon86, Simon87]. Ce sous système offre de riches fonctionnalités en supportant une grande classe d'assertions ainsi que des transactions générales. Les assertions sont manipulées à l'aide d'un langage de haut niveau proche de celui décrit dans [ANSC83]. Pour obtenir de bonnes performances, la méthode utilise deux techniques essentielles : (i) les assertions sont simplifiées et compilées en pré-tests différentiels, (ii) la vérification des assertions compilées est optimisée en spécialisant l'algorithme de contrôle pour trois classes d'assertions. Dans [Simon84], nous avions introduit les algorithmes de vérification des assertions compilées. Dans ce papier, nous présentons en détail la méthode de simplification des assertions par génération de pré-tests différentiels. Les résultats présentés ici sont extraits de [Simon86].

Ce rapport est donc organisé comme suit. La section 2 introduit des définitions préliminaires sur le langage de définition d'assertions que nous utilisons. La section suivante présente tout d'abord ce qu'est une méthode de simplification d'assertions et analyse les travaux existants. Puis le concept de pré-test différentiel et la notion de contrainte compilée sont introduits. La section 4 présente ensuite en détail les règles de décomposition et de simplification d'assertions utilisées. La section 5 décrit les algorithmes généraux de compilation d'une assertion en fonction d'une mise à jour ainsi que les algorithmes particuliers qui ont été implantés dans SABRINA. Finalement, la section 6 est la conclusion

2. DEFINITIONS PRELIMINAIRES

2.1. Calcul relationnel de tuples

Rappelons tout d'abord quelques notions usuelles du modèle relationnel. Un schéma relationnel $\Re$ est un ensemble fini d'attributs $\{A_1, A_2,..., A_n\}$. Soit dom (A_i) le domaine de valeurs sur lequel varie l'attribut A_i. Un *tuple constant* $t = (c_1,..., c_n)$ sur un schéma relationnel $\Re$, est une fonction de $\Re$ vers $\{\text{dom } (A_1) \cup ... \cup \text{dom } (A_n)\}$ telle que pour tout i appartenant à $\{1,..., n\}$, $c_i \in$ dom (Ai). On note dom ($\Re$) l'ensemble de tous les tuples constants sur $\Re$. Une *instance* d'un schéma relationnel $\Re$, (aussi appelée une relation), est un ensemble fini de tuples constants sur $\Re$; on la note I ($\Re$) ou encore R. Un schéma d'une base de données est un ensemble fini de schémas relationnels. Finalement, une instance I sur un schéma relationnel *S*, (encore appelée état d'une base de données), est une fonction totale de *S* telle que pour chaque $\Re$ dans S, I ($\Re$) est une instance sur $\Re$.

Exemple :

Soient PARENT un schéma relationnel {Père, Mére, Enfant} et dom (Père) = dom (Mére) = dom (Enfant) = {noms}.Une instance de la relation PARENT est un ensemble fini de triplets de nom. Donc, I (PARENT) = {(Père = Patrick, Mére = Cati , Enfant = Sarah), (Père = Serge, Mére = Sophie, Enfant = Jérémie), (Père = Robert, Mére = Raymonde, Enfant = Kador)} est une instance de la relation PARENT.

Nous utilisons une version étendue du calcul relationnel de tuples défini dans [Codd71]. Ce calcul est basé sur une logique multi-sorte où les termes sont typés. Il comporte deux types de symboles de prédicats : les prédicats relationnels (unaires) pour lesquels les interprétations correspondent aux instances de schémas relationnels, les prédicats non relationnels (binaires), qui correspondent aux prédicats usuels de comparaison et qui sont définis ci-dessous. Avant d'introduire la définition syntaxique d'une formule, nous nous donnons un alphabet de symboles. Il est composé de symboles de constantes c1,c2, ..., de symboles de variables x, y,... , de noms de fonctions f1,f2,..., auxquels est associée une arité $n \geq 0$, de noms d'attributs A1, A2,..., (qui peuvent être considérés comme des symboles de fonctions d'arité unaire), de noms de prédicats relationnels R1, R2,..., de noms de prédicats non relationnels P1, P2,... , de signes de ponctuation, de connecteurs et de comparateurs logiques. Avec un tel alphabet nous construisons un ensemble d'expressions syntaxiquement bien formées.

Les *termes* sont définis récursivement par :

- une constante ou une variable est un terme,
- si t1,t2,...,tn sont des termes et f un symbole de fonction n-aire, alors f (t1, t2, ...,tn) est un terme.

Nous particularisons un type de fonction comme suit. Si A est un nom d'attribut et x une variable tuple alors A (x) (noté aussi x.A) est un terme représentant la valeur de l'attribut A du tuple x. Nous généralisons également la notion de tuple constant en introduisons la notion de *tuple*: Ainsi, soit $\Re$ $\{A_1, A_2, ...,A_n\}$ un schéma relationnel et t1, t2, ..., tn des termes; si pour tout i dans {1,..., n}, dom (Ai) $\supset$ dom (ti) alors (t1,t2, ..., tn) est un tuple sur $\Re$.

Exemple :

Les constantes Patrick, Sarah, Jérémie sont des termes.
Les attributs x.A, y.B sont des termes.
Les expressions + (x.A, x.B), division (x.A, y.B) sont des termes.
Le tuple (Prénom = Joe, Nom = Dalton , Recherche = x.Prime) est un terme.

Un *prédicat relationnel* est défini comme suit :

- si R est un nom de prédicat relationnel et x une variable tuple alors R (x) est un *prédicat relationnel* (positif) et $\neg$ R (x) est un prédicat relationnel (négatif) .
- si U et V sont deux prédicats relationnels alors U $\wedge$ V est un prédicat relationnel.

Exemple :

PARENTS (x) est un prédicat relationnel.
PARENTS (x) $\wedge$ ANCETRE (y) est un prédicat relationnel.
$\neg$ PINGOUIN (x) est un prédicat relationnel.

OISEAU (x) $\wedge$ ⌉PINGOUIN (y) est un prédicat relationnel.

Une *sous-formule* est maintenant définie récursivement comme suit :

- soient P un nom de prédicat relationnel et t1,t2, ..., tn des termes; alors P (t1, t2, ..., tn) est une sous-formule.
- soient F une sous-formule contenant une variable x et P un nom de prédicat relationnel; alors $(\exists x \in P)$ (F) et $(\forall x \in P)$ (F) sont deux sous-formules. Dans ces deux sous-formules, la variable x est dite quantifiée sur R et $(\exists x \in P)$ (F) , $(\forall x \in P)$ (F) sont appellés *prédicats relationnels quantifiés.*
- soient F et G deux sous-formules; alors $F \wedge G$, $F \vee G$, $\neg F$, $F \Rightarrow G$ sont des sous-formules.

Exemple :

x.A = y.B est une sous-formule.

$(\exists x \in R)$ (x.A = y.B) est une sous-formule.

PARENTS (x) n'est pas une sous-formule.

Une *formule* est alors définie comme suit.

- un prédicat relationnel ou une sous formule est une formule.
- soit U un prédicat relationnel et soit F une sous-formule; alors $U \wedge F$ est une formule.

Dans le cas géneral, une formule du calcul se compose d'une partie qui définit le type de certaines variables tuple et d'une seconde partie connectée conjonctivement à la première qui spécifie une condition que doivent satisfaire ces variables tuples. On dira qu'une variable x est *libre* dans une formule F si F contient une occurence de x non quantifiée. Si x est une variable non libre dans F, on dit que x est une variable *liée* dans F. Une formule dont toutes les variables sont liées est dite *formule fermée.* Une formule dont au moins une variable est libre est dite *formule ouverte.*

Exemple :

- PARENTS (x) est une formule.
- PARENTS (x) $\wedge$ x.Père = Serge est une formule.
- PARENTS (x) $\wedge$ ANCETRE (y) $\wedge$ x.Père = y.Ascendant est une formule.

Nous définissons maintenant l'interprétation d'une formule. Soit D l'ensemble composé de tous les domaines des schémas relationnels et de tous les attributs définis sur un schéma de base de données $S = \{\Re_1, \Re_2, ..., \Re_n\}$. On a D = {dom ($A_1$), ..., dom ($A_m$), dom ($\Re_1$), ..., dom ($\Re_n$)}. Une base de données BD est l'ensemble des instances de tous les schémas relationnels $\Re_1, \Re_2, ..., \Re_n$ (i.e., de tous les prédicats relationnels). On dira que BD est une interprétation. Une formule est alors évaluée de manière classique sur cette interpétation. La formule R (x), où R est un nom de prédicat relationnel, prend la valeur vrai si et seulement si il existe une substitution de la variable x par un tuple constant appartenant à l'interprétation de $\Re$ (i.e., R). La formule $(\forall x \in R)$ (F) s'évalue

comme F (r_1) $\wedge$ F (r_2) $\wedge$ ... $\wedge$ F (r_n) où $r_1, r_2, ..., r_n$ représentent l'ensemble des tuples constants de R. La formule ($\exists$ x $\in$ R)(F) s'évalue comme F (r_1) $\vee$ F (r_2) $\vee$ ... $\vee$ F (r_n).

Le langage de manipulation de contraintes d'intégrité de SABRINA est dans l'esprit de [ANSC83] mais offre plus de généralité. Ce langage est décrit dans [Simon86]. Dans sa forme la plus générale, une contrainte d'intégrité est une assertion du calcul relationnel de tuples c'est à dire une formule fermée. De plus, toutes les assertions sont supposées mises sous *forme prénexe*. Rappelons qu'une assertion est en forme prénexe si elle est de la forme :

(Q1 $r_1 \in R_1$) (Q2 $r_2 \in R_2$) ... (Qn $r_n \in R_n$) (M)

où Qi = $\forall$ ou $\exists$, et M est une sous formule du calcul relationnel de tuples dont les variables $r_1,..., r_n$ sont les seules variables libres dans M. (Q1 $r_1 \in R_1$) (Q2 $r_2 \in R_2$) ... (Qn $r_n \in R_n$) est appelé le *préfixe* de l'assertion tandis que M est appelé *matrice* de l'assertion.

Deux types d'assertions sont distinguées. Les assertions *multi-relation, mono-variable* où deux variables tuple ne peuvent être définies sur la même relation et les assertions *multi-relation multi-variables* où plusieurs variables tuple peuvent être définies sur la même relation.

On peut maintenant introduire la notion de schéma assertionnel d'une base de données.

Définition 1 : Schéma assertionnel

Un *schéma assertionnel* est un couple (S, Σ) où S est un schéma de base de données, et Σ est un ensemble d'assertions du calcul relationnel sur S.

On notera I (S)une instance de S ou encore un "état" de la base de données et on exprimera par (I (S) $\models\Sigma$) le fait que toutes les assertions de Σ sont vraies dans l'état I (S) de la base de données ou encore que I (S) est un *modèle* de Σ.

Un résultat trivial, mais fondamental pour ce qui suivra, est que les assertions considérées ici sont toutes des formules *indépendantes du domaine* [Nicolas82]. Cette propriété signifie que la valeur de vérité d'une formule fermée dont les variables varient sur les relations $R_1, R_2, ..., R_n$, ne dépend que de la valeur de chaque R_i et non pas du domaine associé dom ($\Re_i$). Ceci est fondamental car il est alors possible d'affirmer que si à un instant donné I (S) $\models\Sigma$ et, qu'à un autre instant, seuls des domaines d'attributs ont été modifiés, alors les assertions de Σ sont toujours vraies et on aura encore I (S) $\models\Sigma$.

Précisons finalement la notion de transaction utilisée. Une transaction est constituée d'une séquence de mises-à-jour (ou d'instructions) (e.g. insertion, suppression ou modification) encadrée par un début et une fin de transaction. Chaque instruction met à jour tous les tuples d'une relation qui satisfont une condition donnée exprimée en calcul relationnel. Les requêtes sont donc ensemblistes. Les transactions ont ainsi la généralité des transactions QUEL ou SQL

[Stonebraker76, Astrahan76].

2.2. Base de données jouet

Afin d'illustrer la méthode de simplification décrite dans les sections suivantes, on se donne une base de données composée de quelques relations.

La base modèlise une coopérative vinicole expédiant des vins à des clients qui sont supposés les consommer. Le schéma de la base est représenté ci-dessous. Les buveurs sont caractérisés par un numéro de buveur (attribut NB), un nom, un prénom et un type de buveur (domaine "gros", "moyen", "petit"). Les vins sont décrits par les attributs numéro de vin (NV), cru, millésime et degré. Les vins sont produits par des producteurs caractérisés par un numéro de producteur (NP), un nom, un prénom et une région. Les vins sont bus par les buveurs en une certaine quantité à une date donnée, dans un lieu spécifique. Une certaine quantité de vins a été auparavant récoltée par les producteurs. Les clés des relations sont les attributs soulignés.

BUVEURS (nb, nom, prénom, type)
VINS (nv, cru, millésime, degré)
PRODUCTEUR (np, nom, prénom, région)
ABUS (nb, nv, date, lieu, quantité)
RECOLTE (np, nv, quantité).

3. METHODES DE SIMPLIFICATION DE CONTRAINTES D'INTEGRITE

3.1. Méthodes détectives et préventives.

D'une manière générale, les méthodes assertionnelles de contrôle d'intégrité se classifient entre méthodes détectives et méthodes préventives. Les méthodes basées sur la détection des incohérences utilisent la notion de post-test définie ci-dessous.

Définition : Post-test

Soient (S, Σ) un schéma assertionnel, t une transaction et A une assertion de Σ. Une assertion A' est un *post-test* pour A et t ssi

pour chaque instance I (S) de S et image t (I (S)) de I (S) par t :

$$(I(S) \models A) \Rightarrow ((t(I(S)) \models A) \Leftrightarrow (t(I(S)) \models A'))$$

Cette formule établit que, pourvu que la base de données soit cohérente avant mise à jour, il est nécessaire et suffisant de vérifier le post-test A' pour assurer la cohérence de la base mise à jour vis à vis de A. Un post-test trivial pour A, et une transaction t, est A. Les assertions sont donc évaluées

sur l'état modifié de la base. Cela conduit à des techniques de contrôle d'intégrité lourdes et d'un coût prohibitif [Astrahan76]. Il est clair que l'objectif est de trouver des post-tests dont la vérification coûte moins cher que l'évaluation de l'assertion originale. Une façon de réduire ce coût est de réduire le volume de données accédé pour vérifier une assertion puisque c'est en général le facteur le plus critique dans l'évaluation d'une expression du calcul relationnel. Une méthode originale décrite dans [Nicolas82b] construit de tels post-tests au moment de l'exécution d'une mise à jour t. Les assertions sont exprimées en calcul de domaine. Les transactions sont réduites à des mises à jour d'un seul tuple dans une relation. Par exemple, quand la mise à jour insère un tuple dans une relation R, pour chaque assertion de Σ, la méthode détermine quelles variables peuvent être remplaçées par des constantes issues des valeurs d'attributs du tuple inséré. L'assertion résultant de cette substitution est un post-test. La méthode est aussi capable de détecter quelles assertions sont préservées par la mise à jour sans accéder aucune donnée de la base.

Cependant, les stratégies basées sur des post-tests nécessitent de défaire une transaction lorsque des incohérences sont détectées. Une amélioration de ces stratégies consiste à prévénir l'introduction de ces incohérences dans la base de données de telle sorte qu'une transaction invalide soit abandonnée avant de commettre toute modification dans la base de données. Ceci conduit à la notion de pré-test.

Définition : pré-test

Soient (S, Σ) un schéma assertionnel, t une transaction et A une assertion de Σ. Une assertion A' est un *pré-test* pour A et t ssi

pour chaque instance I (S) de S et image t (I (S)) de I (S) par t :

$(I (S) \models A) \Rightarrow ((t (I (S)) \models A) \Leftrightarrow (I (S) \models A'))$.

Cette formule établit qu'il est nécèssaire et suffisant de vérifier A' dans l'état de la base de données avant mise à jour pour garantir la cohérence de la base modifiée vis à vis de A. La plupart des méthodes récentes de contrôle d'intégrité reposent sur la notion de pré-test [Buneman79, Blaustein81, Gardarin79, Hammer75, Stonebraker75]. Elles préviennent l'introduction d'incohérences dans la base en ne commettant que les transactions qui conduisent à des états cohérents. Des techniques de preuve de théorèmes mécanique peuvent être utilisées pour dériver des pré-tests. Le principe est de prouver qu'une mise à jour, exprimée dans un certain langage, préserve l'ensemble des assertions [Gardarin79, Hammer78, Sheard86]. Ces méthodes ne sont pas très générales mais peuvent être très efficaces lorsqu'elles s'appliquent. Cependant, dans la plupart des réalisations effectuées, le mécanisme de preuve est enclenché à chaque mise à jour et est très lent; (il semble pourtant que l'implantation en Lisp réalisée par T. Sheard dans [Sheard86] utilise des techniques de preuve performantes). Enfin, dans ces méthodes, toutes les mises à jour permises doivent être prédéfinies. L'algorithme de modification de questions de [Stonebraker75], fameux pour son élégance, produit des pré-tests à chaque mise à jour en ajoutant (conjonctivement) l'assertion à la sous formule conditionnant l'exécution de la mise à jour. Cependant, l'algorithme ne

s'applique qu'à des formules du calcul relationnel de tuple dont toutes les variables sont universellement quantifiées. La raison de cette limitation peut être illustrée comme suit. Considérons l'assertion ($\forall$ x $\in$ R) F (x) où F est une formule dont x est la seule variable libre. Une mise à jour de R peut s'écrire ($\forall$ x $\in$ R) (Q (x) $\Rightarrow$ modifier (x)) où Q est une formule dont x est la seule variable libre. Grossièrement, la modification de question consiste à générer la mise à jour ($\forall$ x $\in$ R) ((Q (x) $\wedge$ F (x)) $\Rightarrow$ modifier (x)). Donc x doit être quantifiée universellement.

L'algorithme proposé dans [Blaustein81] est une amélioration de la méthode de [Nicolas82b]. Les transactions sont toujours réduites à des mises à jour mono-tuple, mono-relation. L'algorithme produit des pré-tests à chaque définition d'assertion et pour chaque type de mise à jour (insérer, supprimer). Ces pré-tests sont ensuite contrôlés à chaque mise à jour. La méthode accepte une plus grande classe d'assertions que celle de [Nicolas82b], en fait des assertions multi-relation, mono-variable avec agrégats. Le principe utilisé est la substitution de variables tuple par des constantes issues du tuple mis à jour. A côté de l'importante contribution de ce travail, il reste que la restriction des transactions rend la méthode difficilement utilisable en pratique dans un vrai SGBD. Le travail présenté dans [Henschen84] est une extension des travaux de [Blaustein81] et de [Nicolas82b], basée sur un prouveur de théorème, mais génère des pré-tests dans le même esprit.

Dans la section suivante, nous essayons de combiner la généralité des transactions de mise à jour utilisée dans [Stonebraker75] avec *au moins* la généralité des assertions pour lesquelles des pré-tests peuvent être générés dans [Blaustein81]

3.2. Pré-tests différentiels

Avant d'introduire le concept de pré-test différentiel, d'autres notions sont nécessaires.

Définition 2 : Relations différentielles

Soient $\mathfrak{R}$ un schéma relationnel (*) et t une transaction de mise-à-jour d'une instance R de $\mathfrak{R}$. Alors R^+ et R^- sont appelées les *relations différentielles* de R par t si :

(i) R^+ et R^- sont des instances de $\mathfrak{R}$,

(ii) R^+ contient les tuples insérés par t dans R, et R^- contient les tuples supprimés de R par t.

Notons que dans le cas d'une modification, R peut posséder deux relations différentielles non vides R^+ et R^-.

Remarque : Précisons que la sémantique de la modification d'une relation R consiste d'abord à effectuer R - R^- puis $R^+ \cup (R - R^-)$.

(*) Par la suite, on abrégera l'écriture de $\mathfrak{R}$ $\{A_1,\ldots, A_n\}$ par $\mathfrak{R}$.

Définition 3 : Domaine d'interprétation d'une assertion

Soit A une assertion de Σ utilisant des variables portant sur des instances Ri de $\Re i$ pour i =1 à n. On dira que $R_1,\ldots, R_n$ constituent le *domaine d'interprétation* de l'assertion A et on notera l'assertion A $(R_1,\ldots, R_n)$.

Il est maintenant possible d'introduire la notion de pré-test différentiel.

Définition 4 : Pré-test différentiel

Soient (S, Σ) un schéma assertionnel et t une transaction dont l'effet est soit $t(I(S)) := I \cup I^+$ (t est une insertion) ou $t(I(S)) := I\text{-}I^-$ (t est une suppression) ou $t(I(S)) := I^+ \cup (I - I^-)$ (t est une modification). Soit A $(R_1, ..., R_n)$ une assertion dans Σ et soit A'$(B_1,\ldots, B_m)$ une assertion telle que pour chaque i, $1 \leq i \leq m$, Bi est :

. soit une sous-relation de R_j définie par une expression du calcul relationnel, pour j, $1 \leq j \leq n$

. soit une relation différentielle R^+_j ou R^-_j pour j, $1 \leq j \leq n$, (si R_j est modifié par la transaction t).

A' est un *pré-test différentiel* pour (S, Σ) et t ssi pour chaque instance $I(S)$ de S et image $t(I(S))$ de $I(S)$ par t

$$(I(S) \models A) \Rightarrow [(t(I(S)) \models A) \Leftrightarrow (I(S) \models A')].$$

Les pré-tests différentiels généralisent la notion de pré-test parce qu'ils sont définis sur une structure relationnelle augmentée, c'est-à-dire la base de données et les schémas des relations différentielles.

Moins formellement, le concept de pré-test différentiel poursuit deux buts. Le premier est de permettre un contrôle préventif des assertions lors des mises à jour et donc de transformer l'assertion originale en une expression qui ne dépend plus que des relations avant mise à jour ainsi que des relations différentielles. Cette expression constitue donc une *précondition* à la mise à jour. Le deuxième objectif est de réduire la taille du domaine d'interprétation de l'assertion A'. Ceci apparaît clairement dans la définition ci-dessus puisque des relations Bi, plus petites sont substituées aux relations Ri. Cependant, dire que A' "coûte moins cher" à évaluer que A ou encore que A' est une forme simplifiée par rapport à A, induit une notion de complexité qu'il nous faut préciser.

On prétend que l'évaluation de A' coûte moins cher que l'évaluation de A si le produit cartésien des relations du domaine d'interprétation de A' est plus petit que celui de A. Bien que ce critère semble intuitivement bien fondé, il demeure une approximation car il suppose que la taille des relations différentielles est toujours très petite par rapport à la taille des relations de la base. D'autre part, il n'est pas évident d'établir une comparaison aussi simple lorsque A' est en fait une conjonction ou une disjonctions d'assertions comme on le verra dans les sections suivantes. Par suite, la méthode proposée dans ce chapitre ne constitue qu'une *heuristique d'optimisation* pour

suite, la méthode proposée dans ce chapitre ne constitue qu'une *heuristique d'optimisation* pour évaluer une contrainte.

La méthode de simplification décrite dans ce papier repose, à l'instar de [Blaustein81], sur des règles syntaxiques de transformation des assertions définies par l'usager. Ces règles opèrent une fois pour toutes lors de la définition d'une assertion, pour produire des pré-tests différentiels en fonction de certains types prédéfinis de transaction. Les types de transactions considérés ici sont en fait des mises à jour qui insèrent, suppriment ou modifient un ensemble de tuples d'une seule relation. Par la suite, on notera par : $t(R) := R \cup R^+$ une insertion dans R, $t(R) := R - R^-$ une suppression dans R et $t(R) := R^+ \cup (R - R^-)$ une modification dans R.

On n'utilisera ainsi qu'une version restreinte des pré-tests différentiels. La raison de ce choix est assez simple : il est plus aisé et surtout plus souple de prédéfinir des types de mise à jour plutôt que des séquences de requêtes. En fait, la pré-définition de types de transaction correspond bien à deux situations. La première est la manipulation de transactions pré-compilées [Chamberlin81]. Dans ce cas, lors du stockage d'une transaction, le mécanisme de simplification des assertions est activé et produit un ensemble de pré-tests différentiels stockés avec la transaction. Ceci nécessite cependant que les assertions originales déclarées par l'administrateur soient elles aussi stockées au niveau du schéma. La deuxième situation se produit lorsqu'un utilisateur écrit de manière interactive une transaction. Dans ce cas, les pré-tests différentiels sont construits à "run-time" et la vérification des contraintes ne peut avoir lieu qu'en fin de transaction (ceci était d'ailleurs également vrai dans la première situation). L'inconvénient majeur de cette dernière situation est que si la transaction est non valide, elle doit être entièrement reprise, même si l'erreur intervenait à la première requête.

En conséquence, nous avons respecté deux objectifs : (1) effectuer la simplification des contraintes lors de leur définition et ne stocker que les formes simplifiées, (2) détecter le plus tôt possible une requête invalide dans une transaction, éventuellement interactive.

La généralisation de notre méthode de simplification à des transactions composées d'une séquence de mises à jour nécessitera des mécanismes supplémentaires. En effet, les assertions ne peuvent pas être testées de manière indépendante pour chaque mise à jour de la transaction. La raison en est que certaines contraintes d'intégrité peuvent être temporairement violées durant l'exécution de la transaction, c'est-à-dire avant que toutes les mises à jour de la transaction aient été prises en compte. C'est le cas, par exemple, des dépendances d'inclusion cycliques ou des contraintes référentielles cycliques. Le traitement général d'une transaction, dans le cas où de telles contraintes existent, est discuté dans [Simon86, Simon87].

3.3. Contraintes compilées

Lorsqu'une assertion est définie à l'aide du langage de définition de contraintes d'intégrité, la méthode de simplification est activée et rend en résultat une liste de *contraintes compilées* dont la définition est donnée ci-dessous.

Définition 5 : Contrainte compilée

Soit A une assertion de Σ, une *contrainte compilée* associée à A est un triplet :

(a) ($\Re$,T,E) où $\Re$ est un schéma de relation, T est une insertion ou une suppression dans la relation R et E est un pré-test différentiel pour A et T.

(b) ($\Re'$,T,E) où $\Re \supset \Re'$, T est une modification de la relation R et E est un pré-test différentiel pour A et T.

En fait, on ne gardera pas dans les contraintes compilées l'expression E mais une requête relationnelle permettant à l'aide des critères de la mise à jour de vérifier E. Ce sont finalement les contraintes compilées qui sont stockées dans la métabase du système SABRINA.

Un des critères d'optimisation du contrôle d'intégrité est de minimiser le nombre de contraintes à vérifier lorsqu'une relation est mise à jour. En effet, certaines mises à jour sont sûres de préserver la contrainte tandis que d'autres pourraient la violer et des contrôles sont donc nécessaires. Compte tenu de la définition ci-dessus, cela revient à minimiser le nombre de contraintes compilées à produire pour une assertion donnée. Les théorèmes qui suivent donnent, pour une assertion A et une transaction t, les conditions suffisantes à vérifier pour qu'il n'y ait pas lieu de construire d'assertion compilée, la transaction t ne pouvant conduire à la violation de A. Nous présentons tout d'abord un lemme qui sera utile dans la suite.

Notations :

Soient x et y deux symboles de variables tuple, R et S deux noms de relation et A une assertion, l'expression A [x | y] signifie que toute occurence de y dans A est remplaçée par x, l'expression A[R/S] signifie que toute occurence de S dans un prédicat de relation de A est remplaçée par R et l'expression A [x : R] signifie que toute occurence de x dans A est définie sur la relation R. Enfin, soient f, g1 et g2 trois formules, l'expression f [g1 / g2] signifie que toute occurence de la formule g2 dans f est remplaçée par la formule g1.

Définition 6 : signe d'une formule

Le *signe*, positif ou négatif, d'une formule g occurence d'une formule f est donné par les règles suivantes :

La formule f est positive dans f,

si g est de la forme $\neg$g' alors f est de signe opposé à g dans f

si g est de la forme g1 $\wedge$ g2 ou g1 $\vee$ g2 alors g1 et g2 ont le même signe que g dans f,

si g est de la forme g1 $\Rightarrow$ g2 alors g1 est de signe opposé à g dans f et g2 est de même signe que g dans f,

si g est de la forme ($\forall$ r) g' ou ($\exists$ r) g' alors g' a le même signe que g dans s.

Muni de cette définition, on peut démontrer le lemme suivant :

Lemme 1 :

Soit f une formule du calcul relationnel et g une occurence d'une formule de f. Si g est de signe positif dans f alors (g $\Rightarrow$ g') $\Rightarrow$ (f $\Rightarrow$ f [g '/ g]) est une formule valide. Si g est de signe négatif alors (g $\Rightarrow$ g') $\Rightarrow$ (f [g' / g] $\Rightarrow$ f) est une formule valide.

Preuve :

La démonstration se fait par récurrence. Tout d'abord, supposons que g est positive. Si la formule f est de longueur 1 alors f = g et le résultat est immédiat. Supposons que la propriété soit vraie pour une formule f de longueur inférieure ou égale à n, démontrons la pour une formule f de longueur n+1. Plusieurs cas peuvent se produire.

(1) f est de la forme g1 $\wedge$ g2 ou g1 $\vee$ g2. D'après l'hypothèse, g1 satisfait la propriété et donc pour toute formule g de g1, si g est positif (g $\Rightarrow$ g') $\Rightarrow$ (g1 $\Rightarrow$ g1 [g' / g]). Considérons une formule positive g de f, deux cas se présentent. Si g est une formule de g2 (i.e g = g2) alors on a bien : (g $\Rightarrow$ g') $\Rightarrow$ (g1 $\wedge$ g2 $\Rightarrow$ g1 $\wedge$ g') et de même si f = g1 $\vee$ g2. Maintenant si g est une formule de g1 alors on a bien d'après l'hypothèse : (g $\Rightarrow$ g') $\Rightarrow$ (g1 $\wedge$ g2 $\Rightarrow$ g1 [g' / g] $\wedge$ g2) et de même si f = g1 $\vee$ g2.

(2) Le cas où f est de la forme $\neg$g1 suit immédiatement.

(3) Soit f de la forme (g1 $\Rightarrow$ g2). C'est une combinaison des deux cas précédents.

(4) f est de la forme ($\forall$ r) g1 ou ($\exists$ r) g1. Si g est une formule positive de g1, on a bien (g $\Rightarrow$ g') $\Rightarrow$ (($\forall$ r) g1 $\Rightarrow$ ($\forall$ r) g1 [g' / g]) et de même pour ($\exists$ r) g1.

Le cas où g est négative se démontre de la même façon. ♦

Considérons maintenant une assertion A de la forme (Q r $\in$ R) F (r) où Q est un quantificateur quelconque. Le thèorème suivant donne des conditions suffisantes pour qu'une mise à jour de la relation R préserve ou non l'assertion A. Nous précisons cette dernière notion dans les définitions qui suivent.

Définition 7 : Image d'une assertion par une mise à jour

Soit A une assertion et t une mise à jour d'une relation R. On appelle *image de A par t,* notée A_t, l'assertion A dans laquelle toute expression de la forme (Q r $\in$ R) dans A est remplacée par l'expression (Q r $\in$ t (R))

Définition 8 : Préservation ou affectation d'une assertion par une mise à jour.

Soient t une transaction et A une assertion. On dit que t *préserve* A si A $\Rightarrow$ A_t et on dit que t

affecte A si $A_t \Rightarrow A$.

Thèorème 1 : Soit t une mise à jour d'une relation R et soit A une assertion de la forme (Q r ∈ R) F (r) où Q est un quantificateur. Alors t préserve A si : (i) Q = ∀ et t est une suppression, ou (ii) Q = ∃ et t est une insertion.

Preuve :

A peut s'écrire dans l'une des deux formes suivantes : (∃ r)(R (r) et F (r)) ou bien (∀ r)(R (r) ⇒ F (r)). Si t est une insertion alors R (r) ⇒ $R \cup R^+$ (r). De plus, R (r) est de signe positif dans A : (∃ r)(R (r) et F (r)) et donc par application du lemme 1 on a :

(∃ r)(R (r) et F (r)) ⇒ (∃ r)($R \cup R^+$ (r) et F (r)). Par suite on a bien A ⇒ A_t.

De même, supposons que t soit une suppression. On a (R - R^- (r)) ⇒ R (r). Or R (r) est de signe négatif dans A et donc : (∀ r)(R (r) ⇒ F (r)) ⇒ (∀ r)(R - R^- (r) ⇒ F (r)). Par suite, on obtient bien A ⇒ A_t. ♦

On peut maintenant énoncer le théorème suivant.

Théorème 2 : Soit A une assertion multi-relation mono-variable contenant une expression de la forme g = (Q r ∈ R) F (r) où Q est un quantificateur et soit t une mise à jour de la relation R. Alors, t préserve A si t préserve g et g est de signe positif dans A.

Preuve :

Supposons que g soit de signe positif dans A. Si t préserve g alors g ⇒ g_t et par application du lemme 1 on a A ⇒ A [g_t / g] et A [g_t / g] ≐ A_t car l'assertion est mono-variable. Donc t préserve A.

Remarque : Une particularité d'une formule prénexe est que toute occurence de formule contenue dedans est de signe positif. Si les contraintes d'intégrité étaient sous forme non prénexe alors il faudrait rajouter au théorème ci-dessus la clause que t préserve A si t affecte g et g est de signe négatif dans A. Par exemple soit la contrainte A = ¬ (∃ r ∈ R) F (r) et soit t une suppression dans R, alors il est clair que t préserve A.

Théorème 3 : Soit A une assertion de la forme (Q1 r1 ∈ R1) (Q2 r2 ∈ Rn)... (Qn rn ∈ Rn) (M). Soit t une mise à jour de la relation R. Soient Qe1, Qe2, ... , Qek les quantificateurs portant sur les relations telles que Re1 = Re2 = ... = Rek = R.

(i) si t est une suppression et si pour tout i Qei ≠ ∃ alors A est préservée par t.

Il n'y a donc pas de contrainte compilée du type (ℜ, supprimer, E) associée à A et t.

(ii) si t est une insertion et si pour tout i Qei ≠ ∀ alors A est préservée par t.

Il n'y a donc pas de contrainte compilée du type (ℜ, inserer, E) associée à A et t.

Preuve :

Le résultat s'obtient en appliquant le théorème 2 à A_t successivement à chaque sous formule quantifiée g concernée par t. A chaque étape on a $A \Rightarrow A\,[g_t/g]$. Au total on obtient que $A \Rightarrow A_t$. ♦

La section suivante décrit les règles utilisées pour produire les contraintes compilées dans le cas où une mise à jour affecte une assertion.

4. LES REGLES DE TRANSFORMATION D'UNE ASSERTION

4.1. Introduction

Le principe des règles de transformation peut être décrit comme suit. Considérons une assertion A (R_1, R_2, ... R_n) et une mise à jour t d'une relation R. Le but des règles de transformation est tout d'abord de décomposer A_t en une conjonction ou disjonction d'assertions dans lesquelles toutes les variables sont soit définies sur les relations différentielles R^+ ou R^- soit sur des relations non modifiées par t. Le premier objectif des règles est donc de décomposer une assertion A_t jusqu'à ce qu'un pré-test différentiel soit obtenu. Le second objectif est de simplifier ce pré-test différentiel en utilisant le fait que l'assertion A est supposée vraie avant la mise à jour de manière à obtenir une expression A' plus simple à évaluer que A (en fonction de l'heuristique introduite plus haut). Une caractéristique des règles de décomposition d'une assertion présentées ici est qu'à l'instar de [Blaustein81], les seules informations utilisées pour "casser" A_t sont la forme du préfixe de l'assertion et le fait que l'assertion A était vraie avant mise à jour.

Malheureusement, s'il est possible de démontrer que toute assertion A_t peut toujours être décomposée en un pré-test différentiel, l'expression résultante n'est pas toujours simplifiable. Ceci nous a conduit à isoler une sous-classe d'assertions pour lesquelles un pré-test différentiel peut être obtenu et ce pré-test différentiel est en général plus simple à évaluer que A_t. Plusieurs algorithmes seront alors proposés pour traiter une commande de définition d'assertion appartenant à la sous-classe isolée.

Dans la sous section qui suit nous présentons tout d'abord les règles qui permettent de décomposer une assertion en un pré-test différentiel. Puis, dans les sous sections suivantes, nous introduisons des règles plus fines qui assurent, en général, la dérivation d'un pré-test différentiel simplifié.

4.2. Règles de décomposition d'une assertion

Le premier résultat que nous présentons concerne la *décomposition* d'une formule du type : $(Q\, r \in R)\, (F\,(r))$.

Théorème 4 : Soit A une expression de la forme $(Q\ r \in R)\ (F\ (r))$ où Q est un quantificateur et soit t une mise à jour de la relation R (insertion ou suppression). Alors :

(i) si t est une insertion et $Q = \exists$ alors A se décompose en $A \vee (\exists\ r^+ \in R^+)\ (F\ (r^+))$,

(ii) si t est une insertion et $Q = \forall$ alors A se décompose en $A \wedge (\forall\ r^+ \in R^+)\ (F\ (r^+))$,

(iii) si t est une suppression et $Q = \exists$ alors A se décompose en $(\exists\ r \in R)\ (\neg\ R^-\ (r) \wedge F\ (r))$

(iv) si t est une suppression et $Q = \forall$ alors A se décompose en $(\forall\ r \in R)\ (\neg\ R^-\ (r) \Rightarrow F\ (r))$

Preuve :

On prouve successivement chacun des quatre cas. Dans le premier cas, A s'écrit aussi : $(\exists\ r)(R\ (r) \wedge F\ (r))$. Si t est une insertion alors A devient $(\exists\ r)[(R \cup R^+\ (r) \wedge F\ (r)]$. Par distribution de l'appartenance sur l'union on obtient $(\exists\ r)[(R\ (r) \vee R^+\ (r)) \wedge F\ (r)]$, ce qui donne par distribution du $\wedge$ sur le $\vee$ la formule : $(\exists\ r)[(R\ (r) \wedge F\ (r)) \vee (R^+\ (r) \wedge F\ (r))]$ puis d'après la propriété du quantificateur existentiel on obtient le résultat. Le cas (ii) est analogue à celui-ci. Pour le cas (iii) A s'écrit aussi : $(\exists\ r)(R\ (r) \wedge F\ (r))$. Puisque t est une suppression A devient : $(\exists\ r)(R - R^-\ (r) \wedge F\ (r))$, ce qui s'écrit : $(\exists\ r)((R\ (r) \wedge \neg R^-\ (r)) \wedge F\ (r))$. Par associativité du $\wedge$ on obtient le résultat. Le dernier cas se démontre de manière similaire. A s'écrit $(\forall\ r)(R\ (r) \Rightarrow F\ (r))$; puisque t est une suppression, A devient $(\forall\ r)(R - R^-\ (r) \Rightarrow F\ (r))$, i.e., $(\forall\ r)\ ((R\ (r) \wedge \neg R^-\ (r)) \Rightarrow F\ (r))$. Par distribution du $\Rightarrow$ sur le $\wedge$ on obtient : $(\forall\ r)(R\ (r) \Rightarrow (\neg R^-\ (r) \Rightarrow F\ (r)))$ et on obtient le résultat. ♦

Supposons maintenant qu'une assertion A multi-relation, mono-variable, (cf. définition vue plus haut), contienne une formule f du type $(Q\ r \in R)\ (F\ (r))$. Connaissant le type de la mise à jour, le théorème ci-dessus permet de déduire que A est équivalente à $A\ [g\ /\ f]$. Deux cas génériques se présentent : g est de la forme $f\ \theta\ f'$ (cas (i) et (ii) du théorème), ou g est de la forme $(Q\ r \in R)\ (F\ (r)\ \theta\ f')$, cas (iii) et (iv) du théorème, avec $\theta \in \{\wedge, \vee\}$. Pour chacun de ces cas génériques nous démontrons un théorème qui permet de décomposer A en deux assertions dont l'une est A.

Théorème 5 : Soit A une formule du type $(Q\ r \in R)\ (F\ (r)\ \theta\ G\ (r))$ où $\theta \in \{\wedge, \vee\}$. Alors il existe A' telle que A soit équivalente à $(Q\ r \in R)\ (F\ (r)\ \theta\ A'$.

Preuve :

Plusieurs cas sont à distinguer en fonction du type de connecteur et du quantificateur.

(i) si $Q = \forall$ et $\theta = \wedge$, alors A s'écrit : $(\forall\ r \in R)\ F\ (r) \wedge (\forall\ r \in R)\ G\ (r)$ d'après la propriété du $\forall$. Par suite il suffit de prendre $A' = (\forall\ r \in R)\ G\ (r)$.

(ii) si $Q = \forall$ et $\theta = \vee$, alors A s'écrit : $(\forall\ r \in R)\ (F\ (r) \vee G\ (r))$. Le problème est de trouver A' telle que A équivaut à $(\forall\ r \in R)\ F\ (r) \vee A'$. Or le premier terme disjonctif s'écrit aussi :
$(\forall\ r \in R)\ (F\ (r) \vee (G\ (r) \wedge \neg G\ (r))$ qui s'écrit encore $(\forall\ r \in R)\ ((F\ (r) \vee (G\ (r)) \wedge (F\ (r) \vee \neg G\ (r)))$
Par distribution on a $(\forall\ r \in R)\ (F\ (r) \vee (G\ (r)) \wedge (\forall\ r \in R)\ (F\ (r) \vee \neg G\ (r))$. Donc, on est ramené à trouver A' telle que A équivaut à $[(\forall\ r \in R)\ (F\ (r) \vee (G\ (r)) \wedge (\forall\ r \in R)\ (F\ (r) \vee \neg G\ (r))] \vee A'$. Cette expression est de la forme $(a \wedge b) \vee A'$. Il suffit donc de choisir $A' = a \wedge \neg b$ pour obtenir "a" ce qui est bien le résultat.

(iii) si Q = $\exists$ et $\theta = \wedge$, alors A s'écrit : $(\exists r \in R)(F(r) \wedge G(r))$. On cherche donc A' telle que A équivaut à $(\exists r \in R) F(r) \wedge A'$. Or le premier terme conjonctif s'écrit aussi :
$(\exists r \in R)(F(r) \wedge (G(r) \vee \neg G(r))$ i.e., $(\exists r \in R)((F(r) \wedge G(r)) \vee (F(r) \wedge \neg G(r)))$. Par distribution on a $(\exists r \in R)(F(r) \wedge G(r)) \vee (\exists r \in R)(F(r) \wedge \neg G(r))$. Donc, on est ramené à trouver A' telle que A équivaut à $[(\exists r \in R)(F(r) \wedge G(r)) \vee (\exists r \in R)(F(r) \wedge \neg G(r))] \wedge A'$. Cette expression est du type $(a \vee b) \wedge A'$; il suffit donc de prendre $A' = a \vee \neg b$ pour obtenir "a" et le résultat.
(iv) si Q = $\exists$ et $\theta = \vee$, alors A s'écrit : $(\exists r \in R)(F(r) \vee G(r))$ et par suite d'après la propriété du $\exists$ par rapport au $\vee$ on a : $(\exists r \in R) F(r) \vee (\exists r \in R) G(r)$. D'où le résultat. ♦

Théorème 6 : Soit A une formule contenant une formule f. Pour toute formule f' il existe une formule A' telle que A [f θ f' / f] est équivalente à A θ' A' où A' contient f' et où θ et $\theta' \in \{\vee, \wedge\}$.

Preuve:

La preuve procède par récurrence sur la longueur de A. Tout d'abord pour A de longueur 1 on a A = f et il suffit de prendre A' = f' pour obtenir le résultat. Supposons maintenant la propriété vraie pour A de longueur inférieure ou égale à n, démontrons là pour A de longueur n + 1. Plusieurs cas sont à considérer.
(i) si $A = g1 \wedge g2$. Supposons que f soit dans g1 qui est de longueur n. D'après l'hypothèse de récurrence, on a : g1 [f θ f' / f] équivaut à g1 θ g1'. Donc :
A [f θ f' / f] = g1 [f θ f' / f] $\wedge$ g2, i.e, A [f θ f' / f] = (g1 θ g1') $\wedge$ g2. Par suite :
A [f θ f' / f] = (g1 $\wedge$ g2) θ (g1' $\wedge$ g2) si $\theta = \vee$ ou bien
A [f θ f' / f] = (g1 $\wedge$ g2) θ g1' si $\theta = \wedge$.
(ii) si A = g1 $\wedge$ g2, la démonstration est identique au cas (i).
(iii) si $A = \neg g1$. On a A [f θ f' / f] = $\neg$(g1 [f θ f' / f]), qui donne par hypothèse appliquée à g1 :
A [f θ f' / f] = $\neg$(g1 θ g1') et donc $\neg g1\ \neg\theta\ \neg g1'$.
(iv) si $A = g1 \Rightarrow g2$ alors la démonstration est une combinaison des cas (ii) et (iii).
(v) si A = (Q r $\in$ R) g1, alors A [f θ f' / f] = (Q r $\in$ R) g1 [f θ f' / f] qui donne par hypothèse :
A [f θ f' / f] = (Q r $\in$ R) (g1 θ g1'). D'après le théorème 5 on peut écrire :
A [f θ f' / f] = (Q r $\in$ R) g1 θ g1", d'où le résultat. ♦

Muni des trois théorèmes qui précédent il est possible de décomposer une assertion mono variable en un pré-test différentiel (en fait, le théorème 4 suffirait). La généralisation à une assertion multi-variable ne pose aucun problème. Il suffit de renommer toutes les occurences de la relation mise à jour en des noms de relation différents. On est alors ramené à une mise à jour simultanée de plusieurs relations que l'on traite l'une aprés l'autre dans l'ordre d'apparition au sein de la formule (i.e., de la formule la plus englobante vers la plus englobée). Le problème qui se pose est que toutes les décompositions d'assertion ne conduisent pas à un pré-test différentiel plus simple à évaluer que l'assertion originale. D'une manière générale il n'est de toutes façons pas possible de garantir la

"simplification" puisqu'elle dépend de la base de données et de la nature de la mise à jour. Mais, par une analyse plus fine du préfixe de l'assertion, sous l'hypothèse que les relations différentielles sont toujours trés petites par rapport aux relations de base il est possible d'obtenir une réelle simplification de l'assertion. Le but des sous sections suivantes est précisement d'isoler une classe d'assertions pour laquelle une simplification s'obtient (i) en appliquant les règles de décomposition présentées plus haut et (ii) en utilisant le fait que l'assertion était vraie avant la mise à jour.

4.3. Régles de simplification en Insertion

Dans cette section nous considérons le cas de l'insertion d'un ensemble de tuples R^+ dans une relation R quelconque par une transaction t. On écrira t (R) := $R \cup R^+$. On suppose que dans l'assertion, plusieurs variables sont définies sur la relation R sans aucune restriction concernant la quantification de ces variables.

Deux types de simplification sont possibles pour dériver un pré-test différentiel A' à partir de l'assertion originale A et de t. La première consiste à éliminer de la contrainte A_t, image de A par t, les tests qui sont connus comme trivialement vrais alors que la seconde consiste à éliminer de A_t les tests trivialement faux. Nous donnons par la suite quelques exemples illustratifs.

Exemple 1 :
Soit la contrainte référentielle suivante : "tout tuple de ABUS concerne un buveur de la base"

$A = (\forall\, a \in ABUS)\ (\exists\, b \in BUVEURS)\ (a.nb = b.nb)$

et soit la mise à jour t(ABUS) := $ABUS \cup ABUS^+$. L'image A_t de A par t s'écrira donc :

$A_t = (\forall\, a \in ABUS \cup ABUS^+)\ (\exists\, b \in BUVEURS)\ (a.nb = b.nb)$.

D'aprés ce qui précède, A_t peut être décomposée en deux termes conjonctifs :

$A_t = (\forall\, a \in ABUS)\ (\exists\, b \in BUVEURS)\ (a.nb = b.nb)$

$\wedge\ (\forall\, a^+ \in ABUS^+)\ (\exists\, b \in BUVEURS)\ (a^+.nb = b.nb)$.

Le premier terme n'est rien d'autre que l'assertion A avant mise à jour. Par hypothése, il est donc trivialement vrai. A_t peut donc être réduite à :

$A_t = (\forall\, a^+ \in ABUS^+)\ (\exists\, b \in BUVEURS)\ (a^+.nb = b.nb)$.

A_t est bien un pré-test différentiel pour A et t. Dans cet exemple, la règle que nous avons utilisée est de décomposer A_t en deux termes à partir du prédicat de relation quantifié ($\forall\, a \in ABUS \cup ABUS^+$). La question légitime que l'on peut se poser est : cette simplification est-elle toujours possible ?. Le théorème suivant apporte une réponse.

Théorème 7 : Soit A une assertion de la forme (Q1 r1 $\in$ R1)(Q2 r2 $\in$ Rn)...(Qn rn $\in$ Rn) (M) et soit t une insertion dans R (i.e., t (R) := $R \cup R^+$). Supposons que A contienne un prédicat de relation quantifié de la forme ($\forall\, r_i \in R$).

si pour tout j < i , Qj $\neq \exists$ dans A,

alors on a l'équivalence : $A_t \Leftrightarrow A_t\,[r_i : R] \wedge A_t\,[r_i : R^+]$.

Preuve :

Considérons l'expression : $(\forall r_i \in R \cup R^+)$ F. Ceci s'écrit encore:
$(\forall r_i \in R\ F \wedge \forall r_i \in R^+\ F)$. Puisque tous les quantificateurs qui précédent Qi sont universels, on a :
$A_t = ...(\forall r_{i-1} \in R_{i-1})((\forall r_i \in R)\ F \wedge (\forall r_i \in R^+\ F))$. Cette expression est de la forme :
$A_t = ...(\forall r_{i-1} \in R_{i-1})(f \wedge g)$. En appliquant le théorème 5 on obtient :
$A_t = ...((\forall r_{i-1} \in R_{i-1})\ f) \wedge ((\forall r_{i-1} \in R_{i-1})\ g)$. En répétant l'application du théorème 5 à tous les prédicats de relation quantifiés qui précèdent $(\forall r_i \in R)$ dans A_t on obtient le résultat. ♦

Illustrons ce théorème par un exemple.

Exemple 2 :

Soit la contrainte A définie en français par : "Aucun employé ne peut gagner plus que son patron". La relation EMPLOYE a le schéma EMPLOYE (NOM, DEPART., PATRON, SALAIRE). On a donc :

$A = (\forall a \in EMPLOYE)\ (\forall b \in EMPLOYE)\ ((a.nom = b.patron) \Rightarrow (a.salaire > b.salaire))$.

On appelera M la matrice de cette assertion.

Supposons que t (EMPLOYE) := EMPLOYE $\cup$ EMPLOYE$^+$. D'après le théorème 7, A_t s'écrira :

$A_t = (\forall a \in EMPLOYE)\ (\forall b \in EMPLOYE \cup EMPLOYE^+)\ (M)$

$\wedge\ (\forall a^+ \in EMPLOYE^+)\ (\forall b \in EMPLOYE \cup EMPLOYE^+)\ (M\ [a^+/a])$

Par deux nouvelles applications du théorème 7, on aura :

$A_t = (\forall a \in EMPLOYE)\ (\forall b \in EMPLOYE)\ (M)$

$\wedge\ (\forall a \in EMPLOYE)\ (\forall b^+ \in EMPLOYE^+)\ (M[b^+/b])$

$\wedge\ (\forall a^+ \in EMPLOYE^+)\ (\forall b \in EMPLOYE)\ (M[a^+/a])$.

$\wedge\ (\forall a^+ \in EMPLOYE^+)\ (\forall b^+ \in EMPLOYE^+)\ (M\ [a^+/a, b^+/b])$.

Le premier terme peut être supprimé car trivialement VRAI. Il correspond à l'assertion A avant mise à jour. On a donc remplacé l'évaluation de toutes les combinaisons de tuples dans le domaine produit $(R \cup R^+) \times (R \cup R^+)$ par le domaine produit $(R \times R^+) \cup (R^+ \times R) \cup (R^+ \times R^+)$ où R = EMPLOYE. Une analyse intuitive de la simplification apportée conduit à considérer deux cas :

- si card | R | >> card | R^+ |, il y aura gain en nombre d'E/S et en nombre de comparaisons de tuples.
- si card | R | < card | R^+ |, il y aura toujours gain en nombre de comparaisons de tuples mais perte en nombre d'E/S (lecture des relations opérandes). Le cas extrême étant lorsque card |R| = 1 page et card |R^+| = n >> 1 page.

Cette petite remarque sur le coût d'évaluation d'une assertion, justifie l'hypothèse de départ selon laquelle les méthodes de simplification ne sont efficaces que si les relations différentielles sont petites devant les relations de la base. Notons que cette hypothèse devient d'autant plus forte qu'il y a de variables r_i telles que $R_i = R$, dans le cas du théorème 7.

Une dernière remarque s'impose sur ce théorème. Dans le cas particulier où il n'existe qu'une seule variable r_i telle que $R_i = R$, A_t se décompose mais le premier terme devient toujours trivialement vrai. Donc, dans ce cas, At se réduit à : $A_t [r_i : R^+]$.

On aborde maintenant le second type de simplification applicable à A_t.

Exemple 3 :

Soit la contrainte : "il existe au moins un buveur qui n'ait pas bu".

$A = (\exists\, b \in \text{BUVEURS})\ (\forall\, a \in \text{ABUS})\ (a.nb \neq b.nb)$

Soit t (ABUS) = ABUS $\cup$ ABUS$^+$. L'assertion A_t, image de A par t, s'écrira :

$A_t = (\exists\, b \in \text{BUVEURS})\ (\forall\, a \in \text{ABUS} \cup \text{ABUS}^+)\ (a.nb \neq b.nb)$.

Le théorème 7 ne s'applique pas à A_t puisque b, quantifiée par $\exists$, précède a. Appelons $\overline{\text{BUVEURS}}$, l'ensemble de tous les buveurs qui n'avaient pas bu avant l'insertion dans ABUS. A_t peut se décomposer en :

$A_t = (\exists\, b \in \overline{\text{BUVEURS}})\ (\forall\, a \in \text{ABUS} \cup \text{ABUS}^+)\ (a.nb \neq b.nb)$

$\vee\ (\exists\, b \in \text{BUVEURS} - \overline{\text{BUVEURS}})\ (\forall\, a \in \text{ABUS} \cup \text{ABUS}^+)\ (a.nb \neq b.nb)$.

Le deuxième terme de A_t est trivialement FAUX par définition de $\overline{\text{BUVEURS}}$; il peut donc être éliminé et A_t se réduit à :

$A_t = (\exists\ b \in \overline{\text{BUVEURS}})\ (\forall\, a \in \text{ABUS} \cup \text{ABUS}^+)\ (a.nb \neq b.nb)$.

Il convient donc de déterminer dans quels cas une telle simplification est possible. Auparavant, étant donnée une relation R, nous précisons la signification de $\overline{R}$.

Définition 9 : Combinaison valide d'une assertion

Soit A une assertion contenant n variables r1, r2, ..., rn. On appelle *combinaison valide* de A, noté ϑ (A), l'ensemble minimal des tuples (a1/r1, ... , an/rn) de substitutions des variables r1, ... , rn qui rendent l'assertion A vraie.

Remarque : Pour une même assertion, il existe plusieurs combinaisons valides du fait du choix des substitutions des variables quantifiées existentiellement.

Définition 10 : ensemble redondant

Soit A une assertion contenant un prédicat de relation quantifié ($\exists$ r $\in$ R). On appelle *ensemble redondant* de R pour A, noté $\overline{R}$, l'union de toutes les parties de R appartenant à toutes les combinaisons valides de A.

Exemple 4 :

Soit l'assertion $A = (\exists\, r1 \in R1)\ (\forall\, r2 \in R2)\ (\exists\, r3 \in R3)\ (M)$

Calculons $\overline{R3}$, ensemble redondant de R3 pour A. Supposons que les éléments de R1, R2 et R3 soient :

R1 = {a1, ..., an}, R2 = {b1, ..., bm}, R3 = {c1, ..., cp}.
Considérons une combinaison valide de A. Par exemple :

a1	b1	c1
:	:	|
:	:	|
:	bm-1	c1
:	bm	c2

Alors {c1,c2} est une partie de R3 pour cette combinaison valide. $\overline{R3}$ sera l'union de toutes les parties de R3 de toutes les combinaisons valides de A. Notons que ce calcul de $\overline{R3}$ ne peut s'exprimer sous la forme d'une seule requête algébrique (attention !!!, ce n'est pas : {r3 ∈ R_3 / (∃ r1 ∈ R1)(∀ r2 ∈ R2) (M)}) ?? ... En pratique, il pourra donc être coûteux et complexe de gérer de tels ensembles de données redondants.

On peut maintenant énoncer le théorème suivant.

Théorème 8 : Soit A une assertion de la forme (Q1 r1 ∈ R1)(Q2 r2 ∈ Rn)... (Qn rn ∈ Rn) (M) et soit t une insertion t (R) := R ∪ R^+. Soit (∃ rj ∈ Rj) un prédicat de relation quantifié de A et soit $\overline{Rj}$ l'ensemble redondant de Rj pour A.
si
(i) pour tout i tel que Ri = R on a (Qi ≠ ∀) ou (i > j),
(ii) pour tout i tel que Ri = R on a (Qi ≠ ∃) ou (i = j)
alors A_t est équivalent à A_t ($\overline{R}_j$: R_j)

Preuve :

Soit f = (∃ rj ∈ Rj) F sous formule de A_t. f peut se réécrire comme (∃ rj ∈ $\overline{Rj}$ ∪ $C_{\overline{Rj}}$ (Rj)) F i.e., ((∃ rj ∈ $\overline{Rj}$) F) ∨ (∃ rj ∈ $C_{\overline{Rj}}$ (Rj)) F). f est de la forme g1∨ g2. Plusieurs cas peuvent se présenter selon le prédicat de relation quantifié qui précède f dans A_t.
(i) si on a F' = (∀ r_{j-1} ∈ R_{j-1}) (g1∨ g2) alors d'après le théorème 5, cette expression s'écrit :
(∀ r_{j-1} ∈ R_{j-1}) g1∨ A', avec A' = (∀ r_{j-1} ∈ R_{j-1}) (g1∨ g2) ∧ ¬ ((∀ r_{j-1} ∈ R_{j-1}) (g1∨ ¬g2)).
Discutons la valeur de vérité de A'. Tout d'abord, il faut remarquer que g1 n'est pas nécessairement vrai. Supposons que g1 soit faux. La valeur de vérité de A' suit alors celle de g2. On est donc ramené à étudier si des éléments de $C_{\overline{Rj}}$ (Rj) peuvent rendre F vraie. Deux sous cas sont à considérer.
Premier sous cas : Si tous les prédicats de relation quantifiés de F affectés par la mise à jour sont de la forme (∀ rk ∈ R ∪ R^+) alors g2 ne peut prendre la valeur vrai car tout élément de $C_{\overline{Rj}}$ (Rj) qui satisferait F devrait à fortiori satisfaire F [R / R ∪ R^+], ce qui est contraire à la définition de $C_{\overline{Rj}}$ (Rj).
Deuxième sous cas : supposons que des prédicats de relation quantifiés (∃ rk ∈ R ∪ R^+) figurent dans F. Alors, les éléments de $C_{\overline{Rj}}$ (Rj) peuvent contribuer à rendre l'expression F vraie. Nous

suivants dérivés de la base jouet viticole: BUVEURS {nb, région} et VIN_BU {nv, cru, nb}. Soit l'assertion "Il existe un buveur et un vin bu de la région du buveur qui n'ait jamais été bu par ce buveur". Ceci s'exprime par :

$A = (\exists\, b \in \text{BUVEURS})\ (\exists\, v \in \text{VIN\_BU})\ (\forall\, a \in \text{VIN\_BU})\ (b.\text{région} = v.\text{cru}) \wedge$
$((a.\text{nv} = v.\text{nv}) \Rightarrow (a.\text{nb} \neq b.\text{nb}))$.

Si BUVEURS = {(1, Macon), (2, Bordeaux)} et
VIN_BU = {(10, Macon, 2), (20, Macon, 2)}
alors A est vraie et $\overline{\text{BUVEURS}}$ = {(1, Macon)}.

Soit maintenant t (VIN_BU) := VIN_BU $\cup$ {(10, Macon, 1), (30, Bordeaux, 1)}, alors l'assertion A_t est vraie si l'on prend pour occurence de b le tuple (2, Bordeaux) qui est le seul tuple de BUVEURS à rendre l'assertion vraie.

Plus formellement ceci découle de la propriété de décomposition de $(\exists\, rk \in R \cup R^+)$ donnée par les théorèmes 4 et 5. Or, le deuxième sous cas ci-dessus est éliminé par la seconde condition du théorème 8. Donc, g2 ne peut que prendre la valeur faux et A' devient faux dès que g1 est faux. Par suite, l'expression ci-dessus F' se réduit à $(\forall\, r_{j-1} \in R_{j-1})$ g1 puisque la valeur de vérité ne dépend que de g1.

(ii) si on a $F' = (\exists\, r_{j-1} \in R_{j-1})$ (g1$\vee$ g2) alors d'après le théorème 5, cette expression s'écrit : $F' = (\exists\, r_{j-1} \in R_{j-1})$ (g1$\vee$ g2), i.e., $F' = (\exists\, r_{j-1} \in R_{j-1})$ g1$\vee$ $(\exists\, r_{j-1} \in R_{j-1})$ g2. En appliquant le même raisonnement que ci-dessus, le deuxième terme disjonctif prend toujours la valeur faux. Par suite F' se réduit à $(\exists\, r_{j-1} \in R_{j-1})$ g1.

En généralisant le raisonnement ci-dessus à tous les prédicats de relation quantifiés qui précèdent F' on obtient le résultat du théorème puisqu'aucun de ces prédicats de relation n'est affecté par la mise à jour d'après les hypothèses du théorème 8.

(iii) Le dernier cas à discuter est celui où Rj = R. Dans ce cas, le raisonnement précédent s'applique puisque $f = (\exists\, rj \in R \cup R^+)$ F se décompose en $((\exists\, rj \in R) \vee (\exists\, rj \in R^+))$ F i.e., $(\exists\, rj \in R)$ F $\vee$ $(\exists\, rj \in R^+)$ F et encore $((\exists\, rj \in \overline{Rj})$ F) $\vee$ $(\exists\, rj \in C_{\overline{Rj}}(Rj))$ F) $\vee$ $((\exists\, rj \in R^+)$ F). En appliquant ce qui précède aux deux premiers termes disjonctifs et en reitérant le raisonnement sur l'expression résultante, on obtient le résultat. ceci achève la démonstration. ♦

A titre d'exemple, nous illustrons l'importance de l'hypothèse (i) dans le théorème précédent. Soit la contrainte :

$A = (\forall\, a \in \text{ABUS})\ (\exists\, b \in \text{BUVEURS})\ (a.\text{nb} = b.\text{nb})$.

Si l'assertion A_t associée est par exemple :

$A_t = (\forall\, a \in \text{ABUS U ABUS}^+)(\exists\, b \in \text{BUVEURS})(a.\text{nb} = b.\text{nb})$

Cette assertion ne peut trivialement pas se réduire à :

$A_t = (\forall\, a \in \text{ABUS U ABUS}^+)(\exists\, b \in \overline{\text{BUVEURS}})\ (a.\text{nb} = b.\text{nb})$.

Illustrons maintenant ce théorème à l'aide d'un exemple :

Exemple 5 :

Soit l'assertion A : "il existe au moins un buveur n'ayant pas goûté tous les vins dans des réceptions".

$A = (\exists\, b \in \text{BUVEURS})\ (\exists\, v \in \text{VINS})\ (\forall\, a \in \text{ABUS})\ (b.nb = a.nb \Rightarrow a.nv \neq v.nv)$

Soit la mise à jour : $t\ (ABUS) = ABUS \cup ABUS^+$

D'après le théorème 8, A_t sera réduite à :

$A_t = (\exists\, b \in \overline{\text{BUVEURS}})\ (\exists\, v \in \overline{\text{VINS}})\ (\forall\, a \in ABUS \cup ABUS^+)\ (M).$

Remarquons que dans ce cas, les ensembles de tuples $\overline{\text{BUVEURS}}$ et $\overline{\text{VINS}}$ se calculent facilement. L'assertion A_t exprime alors que lors d'insertions de consommations de vins par des buveurs dans ABUS, il suffit de vérifier "qu'il existe encore au moins un buveur, parmi ceux qui n'avaient pas bu tous les vins, n'ayant pas goûté aux vins qui n'avaient pas encore été bus par des buveurs".

Considérons maintenant l'assertion A :

$A = (\exists\, r_1 \in R)\ (\forall\, r_2 \in R)(M)$ et la mise à jour $t\ (R) := R \cup R^+$. On aura :

$A_t = (\exists\, r_1 \in R \cup R^+)\ (\forall\, r_2 \in R \cup R^+)(M).$

D'après le théorème 8, A_t s'écrira donc :

$A_t = (\exists\, r_1 \in R \cup R^+)\ (\forall\, r_2 \in R \cup R^+)(M).$

Nous allons voir que l'introduction de ces ensembles redondants permet de nouvelles optimisations.

Exemple 6 :

Reprenons l'assertion A de l'exemple 3.

$A = (\exists\, b \in \text{BUVEURS})\ (\forall\, a \in \text{ABUS})\ (b.nb \neq a.nb)$

et soit $A_t = (\exists\, b \in \text{BUVEURS})\ (\forall\, a \in ABUS \cup ABUS^+)\ (b.nb \neq a.nb).$

D'après le théorème 8 on a :

$A_t = (\exists\, b \in \overline{\text{BUVEURS}})\ (\forall\, a \in ABUS \cup ABUS^+)\ (a.nb \neq b.nb).$

Or, A_t peut encore être simplifiée en remarquant que A_t peut se réécrire :

$A_t = (\exists\, b \in \overline{\text{BUVEURS}})\ (\forall\, a \in \text{ABUS})\ (M)$

$\wedge\ (\exists\, b \in \overline{\text{BUVEURS}})\ (\forall\, a^+ \in ABUS^+)\ (M\ [a^+/a])$

En effet, pour être correcte, cette transformation doit garantir que M est vérifiée dans chaque terme conjonctif pour au moins un même buveur b de BUVEURS. Par définition de BUVEURS, toute occurence de b satisfaisant le second terme conjonctif satisfait également le premier terme. A_t peut être encore simplifiée en éliminant le premier terme qui est trivialement vrai par hypothèse. Donc on a :

$A_t = (\exists\, b \in \overline{\text{BUVEURS}})\ (\forall\, a^+ \in ABUS^+)\ (M\ [a^+/a]).$

Les cas d'application de cette simplification sont décrits par le théorème suivant qui est une

généralisation du théorème 7.

Théorème 9 :

Soit A une assertion de la forme (Q1 r1 ∈ R1)(Q2 r2 ∈ Rn)... (Qn rn ∈ Rn) (M) et soit t une insertion t (R) := R ∪ R^+. Soit (∀ ri ∈ R) un prédicat de relation quantifié de A.

Si

(i) Q1 = ∃ et le préfixe de l'assertion A est de la forme (∃ r1 ∈ R1) ∀* (∀ ri ∈ R)... (où ∀* signifie qu'il existe zéro ou un nombre quelconque de ∀ précédant rei mais pas de quantificateurs existentiels),

(ii) R1 est un ensemble redondant pour A,

alors $A_t \Leftrightarrow A_t[ri : R] \wedge A_t[ri : R^+]$.

Preuve :

Soit f = (∀ ri ∈ R ∪ R^+) F, une sous formule de A_t. Alors f s'écrit : (∀ ri ∈ R) F ∧ (∀ ri ∈ R^+) F

En appliquant le théorème 5 à tous les prédicats de relation universellement quantifiés qui précèdent f dans A_t d'après l'hypothèse (i), on obtient :

A_t = (∃ r1 ∈ R1)(G [ri : R] ∧ G [ri : R^+]) et d'après l'hypothèse (ii) :

A_t = (∃ r1 ∈ R1)(G [ri : R] ∧ G [ri : R^+]). Par une nouvelle application du théorème 5 on a : A_t = (∃ r1 ∈ R1) G [ri : R]

∧ [((∃ r1 ∈ R1) G [ri : R] ∧ G [ri : R^+]) ∨ ¬((∃ r1 ∈ R1) G [ri : R] ∧ ¬G [ri : R^+])]. Par définition de R1, G [ri : R] subsume G [ri : R^+] et le second terme conjonctif se réduit à : (∃ r1 ∈ R1) (G [ri : R^+]) ∨ ¬((∃ r1 ∈ R1) ¬(G [ri : R^+])), c'est à dire (∃ r1 ∈ R1) (G [ri : R^+]) ♦

Remarque :

En particulier si ri est la seule variable portant sur R dans A, alors A_t peut être encore simplifiée en : A_t [R^+ / R] puisque le premier terme conjonctif A_t [ri : R] est trivialement vrai par hypothèse.

Nous discutons maintenant l'importance des deux conditions du théorème 9 en les illustrant par des exemples.

Discussion :

Supposons que, dans le théorème 9, Q1 soit quelconque et que Q2 = ∃. Tous les autres quantificateurs précédant Qi restent inchangés. Q2 est alors soit précédé de ∀, soit de ∃. Montrons que dans ces deux cas, la décomposition est en général impossible.

• Cas où Q1 = ∃

- si R1 est différent de $\overline{R1}$ alors la décomposition est en général impossible comme vu dans le théorème 7.

- si R1 = $\overline{R1}$, montrons un contre exemple. Soient $\overline{R1}$ = {a1, a2} et $\overline{R2}$ = {b1, b2}. Tous les éléments de $\overline{R1} \times \overline{R2}$ ne participent pas à la même combinaison valide de A. Supposons qu'il n'existe qu'une seule combinaison valide de A et que les éléments de $\overline{R1} \times \overline{R2}$ participant aux tuples de ϑ (A) soient : (a1, b1) et (a2, b2). Si on applique la décomposition de $\forall$ ri $\in$ (R $\cup$ R^+) on aurait $A_t = A_t$ [ri : R] $\wedge$ A_t [ri : R^+].

Supposons que (a1, b2) soit le seul élément de $\overline{R1} \times \overline{R2}$ rendant le second terme conjonctif vrai dans ϑ (A_t). Par définition, les seuls éléments de $\overline{R1} \times \overline{R2}$ qui participent à rendre valide le premier terme conjonctif sont un sous-ensemble de {(a1, b1), (a2, b2)}. Supposons que ce sous-ensemble ne soit pas vide. Alors, l'assertion A_t serait évaluée vraie alors que la substitution (a1 / r1, b2 / r2) n'appartient à aucun tuple de ϑ (A) et donc rend à fortiori le premier terme conjonctif faux.

Le seul moyen de rendre possible la décomposition serait de conserver explicitement et de manière redondante dans le systéme, tous les éléments de $\overline{R1} \times \overline{R2}$ dans ϑ (A). Nous rejetterons cette possibilité car un tel ensemble d'éléments coûterait bien trop cher à maintenir lors des mises à jour. D'un autre côté, le recalcul de cet ensemble d'éléments, à chaque vérification de la contrainte, devient prohibitif par rapport au coût de l'assertion A_t originale.

• Cas où Q1 = $\forall$

La discussion est identique au cas où Q1 = $\exists$

Supposons maintenant que Q1 = $\exists$ et que tous les quantificateurs Q2 à Qi ne soient pas universels. Soit Qk = $\exists$ avec 1 < k < i un tel quantificateur. On est alors ramené à la situation précédente où il existe une séquence de prédicats de relation quantifiés du type : ($\forall$ r1 $\in$ R1)($\exists$ r2 $\in$ R2) précédant (Qi ri$\in$ R). Ceci achève la discussion. ♦

Notons pour finir que les conditions d'application des Théorèmes 7 et 9 montrent que les assertions A pour lesquelles une simplification de A_t est possible sont celles qui ont un préfixe de la forme : $\exists$ $\forall$* $\exists$*. Par ailleurs, cette forme de préfixe correspond au cas où l'on sait bien calculer $\overline{R}$ pour le premier quantificateur existentiel.

Considérons maintenant le cas des variables quantifiées existentiellement affectées par une insertion.

Exemple 7 :

Soit la contrainte A = ($\exists$ r1 $\in$ R1) ($\forall$ r2 $\in$ R1) (M) et soit la mise à jour t (R1) = R1 $\cup$ $R1^+$. On aura :

A_t = ($\exists$ r1 $\in$ R1 $\cup$ $R1^+$) ($\forall$ r2 $\in$ R1 $\cup$ $R1^+$) (M)

A_t peut être décomposée en :

(1) A_t = ($\exists$ r1 $\in$ R1) ($\forall$ r2 $\in$ R1 $\cup$ $R1^+$) (M)

$\vee$ ($\exists$ $r1^+$ $\in$ $R1^+$) ($\forall$ r2 $\in$ R1 $\cup$ $R1^+$) (M [$r1^+$ | r1])

D'après le théorème 8, on peut écrire :

$A_t = [(\exists\, r1 \in R1)\ (\forall\, r2 \in R1 \cup R1^+)\ (M)$

$\vee\ [\exists\, r^+1 \in R^+1)\ (\forall\, r2 \in R1 \cup R^+1)\ (M\ [r^+1 \mid r1]\ )]$

et d'après le théorème 9, A_t devient :

$A_t = (\exists\, r1 \in R1)\ (\forall\, r^+2 \in R^+1)\ (M)$

$\vee\ (\exists\, r^+1 \in R^+1)\ (\forall\, r2 \in R1 \cup R1^+)\ (M)$

Cette simplification de A_t a été rendue possible par la décomposition initiale de A_t sous la forme (1). Cette dernière simplification est l'objet du théorème suivant.

Théorème 10:

Soit A une assertion de la forme (Q1 r1 ∈ R1)(Q2 r2 ∈ Rn)... (Qn rn ∈ Rn) (M) et soit t une insertion t (R) := $R \cup R^+$. Soit ($\exists\, r_i \in R$) un prédicat de relation quantifié de A.

Si pour tout $j < i$, $Qj \neq \forall$

alors $A_t \Leftrightarrow A_t [r_i : R] \vee A_t [r_i : R_i^+]$

Preuve :

Appelons $f = (\exists\, r_i \in R \cup R^+)\ F$ une sous formule de A_t. f s'écrit encore : $(\exists\, r_i \in R \vee \exists\, r_i \in R^+)\ F$, i.e., $(\exists\, r_i \in R)\ F \vee (\exists\, r_i \in R^+)\ F$. En appliquant succesivement le théorème 5 à tous les prédicats de relation quantifiés (existentiellement) qui précèdent f dans A_t on obtient le résultat. ♦

4.4. Règles de simplification en suppression

Dans cette section, des règles de simplification d'une assertion A_t en présence d'une suppression sont identifiées. Le premier type de simplification applicable a pour objectif de réduire la partie de variable universellement quantifiée. Considérons l'exemple suivant.

Exemple 8 :

Soit la contrainte référentielle suivante :

$A = (\forall\, a \in ABUS)(\exists\, v \in VINS)\ (a.nv = v.nv).$

Supposons une mise à jour t (VINS) := VINS-$VINS^-$. L'assertion A_t s'écrira :

$A_t = (\forall a \in ABUS)\ (\exists\, v \in VINS\text{-}VINS^-)\ (a.nv = v.nv).$

Il apparaît clairement qu'il n'est pas utile de vérifier tous les tuples de ABUS. Il suffit de se limiter aux seuls tuples de ABUS qui pourraient être affectés par la suppression de tuples dans la relation VINS. Les tuples affectés de ABUS appartiennent à l'ensemble

$ABUS' = \{t \in ABUS / A\ [t / a,\ v^- / v,\ v^- : VINS^-]\}.$

A_t peut donc s'écrire, d'après le théorème 7 :

(1) $(\forall a \in ABUS\text{-}ABUS')\ (\exists\, v \in VINS\text{-}VINS^-)\ (a.nv = v.nv)$

$\wedge\ (\forall\, a \in ABUS')\ (\exists\, v \in VINS\text{-}VINS^-)\ (a.nv = v.nv).$

Par définition de ABUS', le premier terme conjonctif est trivialement vrai. A_t se réduit donc à :

$A_t = (\forall\, a \in \{t \in ABUS\ /(\exists\, v^- \in VINS^-)\ (t.nv = v^-.nv)\})(\exists\, v \in VINS\text{-}VINS^-)\ (a.nv =$

v.nv).
où le premier prédicat de relation est un prédicat de relation généralisé du calcul relationnel.

Remarquons qu'A_t n'est pas un pré-test différentiel puisqu'il dépend de VINS-VINS$^-$. La transformation suivante permet de se ramener à un pré-test différentiel. ($\exists$ v $\in$ VINS-VINS$^-$) se réécrit d'après le théorème 4 en :

($\exists$ v $\in$ VINS) ($\forall$ v$^-$ $\in$ VINS$^-$) (v.nv $\neq$ v$^-$.nv).

Par suite A_t devient :

A_t = ($\forall$ a $\in$ ABUS')($\exists$ v $\in$ VINS)($\forall$ v$^-$ $\in$ VINS$^-$) ((v.nv $\neq$ v$^-$.nv) $\wedge$ (a.nv = v.nv))

Examinons intuitivement le gain apporté par cette décomposition dans l'évaluation de A_t L'évaluation de ABUS' réclame une jointure entre ABUS et VINS$^-$. Deux jointures sont ensuite nécessaires : VINS |×| VINS$^-$ et ABUS' |×| VINS. La décomposition permet donc de ramener l'évaluation de A_t à trois jointures dans lesquelles une des deux relations opérandes est supposée très petite par rapport aux relations de la base. Le gain par rapport à l'évaluation des jointures ABUS |×| VINS et VINS |×| VINS$^-$ nécessaires pour évaluer l'assertion A_t originale, dépend donc étroitement de la taille de VINS$^-$ ainsi que de la sélectivité de la jointure ABUS |×| VINS$^-$. De ce point de vue, la règle de simplification utilisée ne représente qu'une heuristique d'optimisation.

Une deuxième simplification est apportée à travers l'utilisation du prédicat de relation généralisé : ($\forall$ a $\in$ ABUS'). Dans le cas particulier où aucun des vins supprimés n'a été bu, ABUS' est vide et par suite A_t est trivialement évaluée à vrai. L'évaluation de ABUS' constitue donc un critère d'arrêt dans la vérification de A_t. Le problème est de savoir si cette simplification est toujours possible.

Deux questions se posent :

(a) Dans quels cas la décomposition (1) ci-dessusdu prédicat de relation universellement quantifié est-elle possible ?

(b) Tous les prédicats de relation existentiellement quantifiés du type ($\exists$ v $\in$ VINS-VINS$^-$) peuvent-ils être utilisés pour effectuer la décomposition (1) en utilisant le calcul de ABUS' ?

La réponse à la question (a) est donnée par les conditions d'application des théorèmes 7 et 9. La réponse à la question (b) nécessite une analyse plus fine.

Exemple 9 :

Soit l'assertion A :

A = ($\forall$ r1 $\in$ R1) ($\exists$ r2 $\in$ R) ($\forall$ r3 $\in$ R3) ($\exists$ r4 $\in$ R) (M)

et soit la mise à jour t (R) := R - R$^-$.

Montrons d'abord que la transformation (1) de l'exemple précédent conduit à une incohérence lorque l'on calcule l'ensemble des tuples affectés comme pour ABUS'. Pour celà, calculons R'1,

c'est-à-dire l'ensemble des tuples de R1 affectés par la suppression de tuples de R. R'1 est l'union de deux ensembles. Le premier contient les tuples de R1 affectés par la suppression d'occurences de r2 tandis que le second contient les tuples de R1 affectés par la suppression d'occurences de r4. Le premier ensemble se calcule comme :

(1) $\{t \in R1 / A [t / r1, r2^- / r2, r2^- : R^-]\}$.

Par contre, le second ensemble ne peut se calculer comme :

(2) $\{t \in R1 / A [t / r1, r4^- / r4, r4^- : R^-]\}$.

En effet, soit ϑ (A) l'unique combinaison valide de A représentée par :

a1	b1	c1	b1
a1	b1	c2	b2
R1	R	R3	R

Soit t (R) := R - {b2}. Le calcul de l'ensemble de tuples de R1 affectés par la suppression de cette occurence de r4 conduit, d'après (2) à l'ensemble vide. Comme le calcul des tuples de R1 affectés par la suppression d'occurences de r2 donne aussi, d'après (1), l'ensemble vide, l'application de la transformation de A_t entraîne que t est valide, ce qui est faux.

Pour comprendre la manière correcte de calculer, dans ce cas, les tuples de R1 affectés par les r4 supprimés, on utilise le diagramme suivant représentant la forme des tuples dans ϑ (A). On suppose : R1 = {a1, ... , an}, R = {b1, ... , bp} et R3 = {c1, ... , cm}.

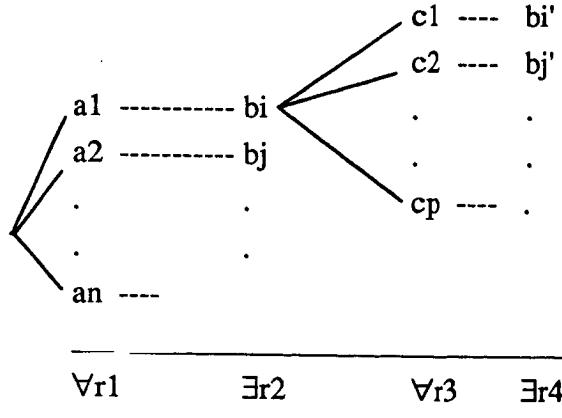


Fig. IV.1 : Diagramme d'une combinaison valide de A

Ainsi dans le diagramme de la figure IV.1 ci-dessus, pour chaque occurence de r1, il existe une occurence de r2 (les occurences de r2 peuvent être toutes différentes deux à deux), telle que chacune d'elle soit reliée à toutes les occurences de r3 dont chacune est à son tour reliée à au moins une occurence de r4. Toutes les combinaisons de tuples ainsi constituées appartiennent à ϑ (A). Elles constituent l'ensemble minimum des combinaisons de tuples permettant de rendre A vraie. Il

peut bien sûr exister d'autres combinaisons valides (en fonction du choix des occurences des variables existentiellement quantifiées).

Par suite, le calcul des "ai" affectés par la suppression de tuples bj suit les étapes :

(1) Trouver les ck affectés par la suppression d'occurences de R pour toutes les compositions d'occurences de r1 et r2 appartenant à ϑ (A).

(2) Trouver, à partir des ck de l'étape (1), les ai affectés.

Ce calcul devient rapidement complexe et inutilisable pour espérer trouver une forme simplifiée de A_t. On se limitera donc aux types d'assertions pour lesquelles le calcul "des tuples affectés" se fait simplement comme dans l'exemple 8 précédent. Les cas défavorables sont en fait ceux où une variable quantifiée par $\forall$ est suivie par un "$\exists$" puis un "$\forall$". Remarquons simplement que la situation suivante illustrée par le diagramme de la figure IV.2 n'est pas gênante.

On suppose que R_1 est {a1, a2, ..., an}, R_2 = {c1, c2, ..., cp} et R_3 = {b1, b2, ..., b_m} et que l'assertion est A = ($\forall$ r1 $\in$ R1) ($\forall$ r2 $\in$ R) ($\exists$ r3 $\in$ R3)(M).

Les tuples r1 affectés par la suppression d'occurences de r3 se calculent simplement par :

$R'_1 = \{t \in R1 / A [r_1 : t, r_3^- / r3, r_3^- : R_3^-]\}$.

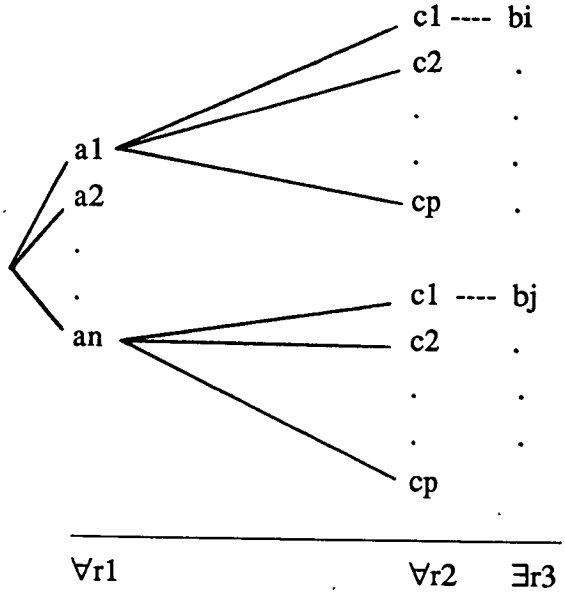


Fig. IV.2 : Diagramme d'une combinaison valide de A

On peut donc énoncer le théorème suivant :

Théorème 11 : Soit A une assertion de la forme (Q1 r1 $\in$ R1)(Q2 r2 $\in$ Rn)... (Qn rn $\in$ Rn) (M) et soit t une suppression t (R) := R - R^-. Soit ($\forall$ ri $\in$ Ri) un prédicat de relation quantifié de A.

Si (i) le préfixe de A est du type $\forall^*\forall$ri,

(ii) pour tout j > i, tel que $Q_j = \exists$ et $Rj = R$, il n'existe pas de séquence $\exists\forall$ entre Qi et Qj,

alors : $A_t \Leftrightarrow A_t [\cup R_i' / R_i]$ où :

$R_i' = \{t \in Ri / A [r_i : t, r_j^- / r_j, r_j^- : R^-]\}$ pour chaque j > i tel que $Rj = R$ et $Q_j = \exists$.

Preuve :

Posons $f = (\forall ri \in Ri)$ F, une sous formule de A. f s'écrit aussi :
$f = (\forall ri \in C_{R'i} (Ri) \cup R_i')$ F, i.e., $f = (\forall ri \in C_{R'i} (Ri) \wedge \forall ri \in R_i')$ F et donc $f = (\forall ri \in C_{R'i}$ (Ri)) F $\wedge$ $(\forall ri \in R_i')$ F. $C_{R'i}$ (Ri) désigne ici le *complémentaire* dans Ri de R_i'. En appliquant le théorème 5 à f dans A_t d'après l' hypothèse (i) du théorème, on obtient : $A_t \Leftrightarrow A_t [r_i : C_{R'i} (Ri)] \wedge A_t [r_i : Ri']$.

Nous allons montrer que le premier terme conjonctif dans l'expression ci-dessus prend toujours la valeur vraie. Rappelons auparavant les hypothèses de travail que nous aurons à utiliser au cours de la démonstration en dehors des hypothèses du théorème lui-même. l'hypothèse H1 est donnée par la définition de l'ensemble R_i'. Cette hypothèse exprime que : $A [r_i : C_{Ri} (Ri'), r_j : R^-]$ est faux. La seconde hypothèse, H2, exprime que A était vraie avant mise à jour.

Soit j le plus petit entier tel que $Qj = \exists$ et $Rj = R$, posons $f = (\exists rj \in R - R^-)$ F, une sous formule de A_t. D'après le théorème 4, f s'écrit : $f = (\exists rj \in R) (\neg R^-(rj) \wedge F)$, ce qui s'écrit encore d'après le théorème 5 : $f = (\exists rj \in R)$ F $\wedge A_1$ où $A_1 = (\exists rj \in R) (\neg R^- (rj) \wedge F) \vee \neg ((\exists rj \in R) (R^-$ (rj) $\wedge$ F)). Pour démontrer que $A_t [r_i : C_{R'i} (Ri)]$ est toujours vrai, nous allons procéder par récurrence sur la longueur du préfixe compris entre Qi et Qj.

Si la longueur du préfixe est 1 deux cas sont à distinguer. Le quantificateur qui précède Qj est soit $\exists$ soit $\forall$.

Cas 1 : $Q_{j-1} = \exists$

Posons $f = (\exists r_{j-1} \in R_{j-1})(a1 \wedge A_1)$ avec $a1 = (\exists rj \in R)$ F. D'après le théorème 5 on a :
$f = (\exists r_{j-1} \in R_{j-1})$ a1 $\wedge A_2$ où $A_2 = (\exists r_{j-1} \in R_{j-1})(a1 \wedge A_1) \vee \neg (\exists r_{j-1} \in R_{j-1})(a1 \wedge \neg A_1)$.
Itérons cette expression au rang i. On pose $f = (\forall ri \in C_{R'i} (Ri))(a2 \wedge A_2)$ avec :
$a2 = (\exists r_{j-1} \in R_{j-1})$ a1. D'après le théorème 5 on a $f = (\forall ri \in C_{R'i} (Ri))$ a2 $\wedge$ $(\forall ri \in C_{R'i} (Ri))$ A_2). Puis en itérant au rang 1, d'après l'hypothèse (i) du théorème 11, on a : $A_t [r_i : C_{R'i} (Ri)]$ équivaut à $A_t [r_i : C_{R'i} (Ri), rj : R] \wedge G$ où $G = (\forall r1 \in R1) \ldots (\forall ri \in C_{R'i} (Ri))$ A_2. On est donc ramené à montrer que ces deux termes sont toujours vrais. Commençons par G. Nous allons raisonner par l'absurde. Supposons que G soit faux et montrons que l'on aboutit toujours à une contradiction.

Notons tout d'abord que A2 s'écrit : $\alpha \vee \beta$. Par suite, dire que G est faux implique que :
$(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i} (Ri))$ α est faux et que
$(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i} (Ri))$ β est faux.

cas 1.1 : si $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i} (Ri))$ α est faux.

Remplaçons α par sa valeur. On a : $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i} (Ri))(\exists r_{j-1} \in R_{j-1})(a1 \wedge A_1)$ est faux. Si a1 = $(\exists rj \in R)$ F est faux alors il y a contradiction avec l'hypothèse H2. En effet, tout d'abord tous les éléments de $C_{R'i}$ (Ri) sont aussi des éléments de Ri. De plus, supposons qu'il existe d'autres prédicats de relation quantifiés dans F tels que Rk = R avec k > j. S'il existe un seul

$Qk = \forall$ alors tous les autres prédicats de relation quantifiés portant sur R sont tels que $Qk = \forall$ par application de l'hypothèse (ii) du théorème 11. Auquel cas, si A était vraie avant mise à jour c'est à fortiori vrai aprés mise à jour pour $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))(\exists r_{j-1} \in R_{j-1})(\exists rj \in R)$ F. S'il existe $k > j$ tel que $Qk = \exists$ et $Rk = R$, alors toujours d'après l'hypothèse (ii) du théorème 11, tous les quantificateurs compris entre Qj et Qk sont existentiels. Par suite, si A était vraie avant mise à jour c'est encore à fortiori vrai pour $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))(\exists r_{j-1} \in R_{j-1})(\exists rj \in R)$ $\exists^* (\exists rk \in R)$ F'.

Soit maintenant r° une occurence de r_{j-1} satisfaisant l'assertion supposons que $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))$ $A_1[r^\circ / r_{j-1}]$ est toujours faux. En remplaçant A1 par sa valeur, on a :

[1] : $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))((\exists rj \in R)(\neg R^-(rj) \wedge F[r^\circ / r_{j-1}])$

$\vee \neg ((\exists rj \in R)(R^-(rj) \wedge F[r^\circ / r_{j-1}]))$.

En particulier on doit avoir : $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))(\neg ((\exists rj \in R)(R^-(rj) \wedge F[r^\circ / r_{j-1}])))$ est faux. Or, $(\exists rj \in R)(R^-(rj) \wedge F[r^\circ / r_{j-1}])$ s'écrit : $(\exists rj \in R^-)(F[r^\circ / r_{j-1}])$ et par suite si l'expression précédente est fausse alors il y a contradiction avec l'hypothèse H1. En effet, la discussion précédente sur l'existence de $k > j$ tels que $Rk = R$ s'applique de nouveau et confirme à fortiori l'hypothèse H1. Ce cas suffit à montrer que si l'expression [1] était fausse il y aurait nécessairement une contradiction.

cas 1.2 : si $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))$ β est faux.

En remplaçant β par sa valeur, on a :

$(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))(\neg (\exists r_{j-1} \in R_{j-1})(a1 \wedge \neg A_1))$ est faux. Soit r° une occurence de r_{j-1} satisfaisant a1, en remplaçant A1 par sa valeur, on doit avoir :

$(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))((\exists rj \in R)(\neg R^-(rj) \wedge F[r^\circ / r_{j-1}]) \vee \neg ((\exists rj \in R)(R^-(rj) \wedge F[r^\circ / r_{j-1}])))$ est faux. En particulier :

$(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))(\neg ((\exists rj \in R)(R^-(rj) \wedge F[r^\circ / r_{j-1}])))$ est faux. Or ceci est contraire à l'hypothèse H1.

Ceci achève la démonstration que G prend toujours la valeur vraie. Reste à montrer que $A_t[r_i : C_{R'i}(Ri), rj : R]$ est toujours vrai. Ceci découle directement de l'hypothèse H2 avec la même discussion que dans le cas 1.1 pour les prédicats de relation quantifiés qui suivent Qj.

Cas 2 : $Q_{j-1} = \forall$

Posons $f = (\forall r_{j-1} \in R_{j-1})(a1 \wedge A_1)$ avec $a1 = (\exists rj \in R)$ F. Ceci s'écrit :

$(\forall r_{j-1} \in R_{j-1})$ a1 $\wedge$ $(\forall r_{j-1} \in R_{j-1})$ A_1. En itérant jusqu'au rang 1, on obtient qu'A_t équivaut à $A_t[ri : C_{R'i}(Ri), rj : R] \wedge (\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))(\forall r_{j-1} \in R_{j-1})$ A_1 Le premier terme conjonctif prend toujours la valeur vraie d'après l'hypothèse H2. Etudions le second terme. Il s'écrit en remplaçant A1 par sa valeur :

[1] : $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))(\forall r_{j-1} \in R_{j-1})((\exists rj \in R)(\neg R^-(rj) \wedge F) \vee \neg ((\exists rj \in R)$ $(R^-(rj) \wedge F)))$. Or si l'expression $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))(\forall r_{j-1} \in R_{j-1})(\exists rj \in R)$ $(\neg R^-(rj) \wedge F)$ était fausse ce serait contraire à l'hypothèse H2. Par conséquent, l'expression [1] prend toujours la valeur vraie et ceci achève la démonstration pour la longueur du préfixe égale à 1.

Supposons maintenant la propriété vraie pour une longueur de préfixe au moins égale à n,

démontrons là pour n + 1. Deux cas peuvent se produire. Le quantificateur rajouté dans la séquence est soit existentiel soit universel. Le cas universel est trivial à traiter et la discussion est analogue au cas 2 ci-dessus. Si l'on rajoute un quantificateur existentiel, alors écrivons l'hypothèse de récurrence. Elle exprime que $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))\forall^* (\exists rk \in Rk)(ak \wedge Ak)$ est toujours vrai pour un k donné tel que $k < j$. Si l'on rajoute un quantificateur existentiel, par exemple Qk, on a à évaluer l'expression :
$(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))\forall^* (\exists rk \in Rk)(\exists rk+1 \in Rk+1)(ak+1 \wedge Ak+1)$ qui se réécrira en : $(\forall r1 \in R1) \ldots (\forall ri \in C_{R'i}(Ri))\forall^* (\exists rk \in Rk)(ak \wedge Ak)$. On est alors ramené à la discussion du cas 1 ci-dessus. La démonstration se fait de manière identique. ♦

Un deuxième type de simplification est applicable dans le cas d'une suppression. Il consiste à éliminer des tests trivialement faux dans A_t.

Exemple 10 :
Soit la contrainte : "il existe au moins un vin qui n'ait pas été bu" :

$A = (\exists v \in VINS)(\forall a \in ABUS)(a.nv \neq v.nv)$

Soit t (VINS) := VINS-VINS$^-$. On aura :

$A_t = (\exists v \in VINS\text{-}VINS^-)(\forall a \in ABUS)(a.nv \neq v.nv)$.

Il suffit de remarquer que A_t peut se réduire à :

(1) $A_t = (\exists v \in VINS\text{-}VINS^-)(\forall a \in ABUS)(a.nv \neq v.nv)$ par définition de VINS.
De plus, comme dans ce cas VINS s'écrit :
$\{t \in VINS / A[v / t]\}$, A_t se réduit à $A_t = (\exists v \in VINS\text{-}VINS^-)(VRAI)$ où VRAI est un prédicat atomique. Le pré-test différentiel s'écrira donc :

$A_t = (\exists v \in VINS)(\forall v^- \in VINS\text{-})(v.nv \neq v^-.nv)$

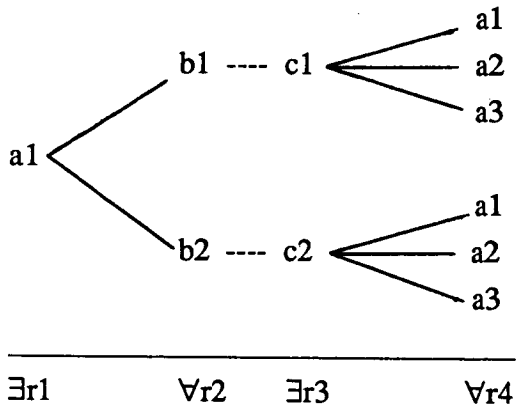
par décomposition du prédicat de relation généralisé.

Plus généralement, est-il toujours possible d'effectuer la transformation (1) ci-dessus ?. Pour répondre à cette question exhibons les cas à problèmes à l'aide de l'exemple suivant:

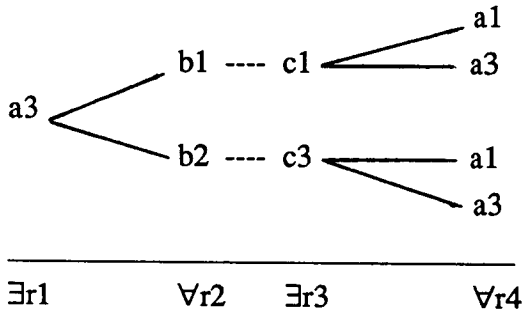
Exemple 11 :
Soit l'assertion A :

$A = (\exists r1 \in R)(\forall r2 \in R2)(\exists r3 \in R3)(\forall r4 \in R)^-(M)$

et la mise à jour t (R) := R-R$^-$
On suppose : R = {a1,a2,a3}, R2 = {b1,b2} et R3 = {c1, ...}.
Soit le diagramme représentant l'ensemble unique des combinaisons valides pour A.



Soit maintenant le diagramme suivant représentant un ensemble de combinaisons de tuples satisfaisant M:



Notons que ce dernier diagramme de combinaisons ne rend pas l'assertion A vraie.

D'après ce qui précède, si la mise à jour t est t (R) := R - {a2}, alors $\bar{R}$ = {a1}. Pourtant A_t ne peut se réduire à :

$A_t = (\exists\, r1 \in \bar{R}\text{-}R^-)\ (\forall\, r2 \in R2) \ldots$

car le terme suivant n'est pas trivialement faux :

$(\exists\, r1 \in (R\text{-}\bar{R}) - R^-)\ (\forall\, r2 \in R2) \ldots$

En effet, dans notre exemple, le second diagramme de combinaisons rend l'assertion A vraie. Ainsi un tuple de $(R\text{-}\bar{R}) - R^-$ contribue à rendre l'assertion vraie puisque $a_3 \in (R\text{-}\bar{R}) - R^-$.

Plus généralement, le seul cas où la transformation sur r1 sera applicable est celui où toutes les variables comprises entre r1 et r4 sont quantifiées par ∀.

Ceci conduit au théorème ci-dessous.

Théorème 12 : Soit A une assertion de la forme (Q1 r1 ∈ R1)(Q2 r2 ∈ R2)... (Qn rn ∈ Rn) (M)
et soit t une suppression t (R) := R - R^-. Soit j tel que $Q_j = \exists$.
pour tout i tel que Qi = ∀ et Ri = R
si le préfixe de l'assertion est de la forme ∃ ∀ * Qi et r_1 porte sur $\bar{R}_1$,

alors : A_t est équivalent à A_t [$\overline{Rj}$ / Rj]

Preuve :

La preuve découle directement des théorèmes 7 et 9. En effet, posons f = ($\forall r_i \in R$) F, une sous formule de A. f s'écrit aussi f = ($\forall r_i \in (R - R^-) \cup R^-$) F, i.e., f = ($\forall r_i \in (R - R^-)$) F $\wedge$ ($\forall r_i \in R^-$) F. En appliquant les théorèmes 7 et 9 à tous les prédicats de relation quantifiés qui précèdent f dans A, on a d'après l'hypothèse du théorème 12 et l'hypothèse que A est vraie :
A [$r_i : (R - R^-)$] $\wedge$ A [$r_i : R^-$] est vrai. De plus, par définition de $\overline{R}_j$, on peut écrire :
A [$r_i : (R - R^-), r_j : \overline{R}_j$] $\wedge$ A [$r_i : R^-, r_j : \overline{R}_j$] est vrai. Donc, chaque terme conjonctif est vrai et en particulier le premier qui correspond à A_t [$\overline{Rj}$ / Rj]. Ce qui démontre le théorème. ♦

4.5. Cas d'une modification.

Dans le cas d'une modification la mise à jour d'une relation R est du type t(R) := $(R - R^-) \cup R^+$. La décomposition de l'assertion A_t s'effectue d'abord à l'aide des théorèmes 5 et 6. On obtient alors plusieurs assertions qui ne dépendent plus que soit de $R - R^-$ soit de R^+. La simplification de ces assertions est alors donnée par les théorémes 10, 11 et 12. En particulier, les théorémes 8 et 12 ne peuvent s'appliquer que si leurs prémisses sont simultanément satisfaites. Ceci permet alors d'éliminer les tests trivialement faux donnés par des occurences de $(R - R^-) \cup R^+$.

4.6. Bilan des régles de simplification

Cette section se propose de résumer les résultats obtenus à l'aide des régles de simplification présentées dans cette section 4. Ce bilan a deux objectifs qui sont d'abord de connaitre les formes syntaxiques de préfixe pour lesquelles des simplifications significatives de l'assertion A_t, image de A par une mise à jour t, sont possibles mais aussi d'isoler une sous classe d'assertions correspondant à des algorithmes de simplification simples à implanter.

Pour résumer les cas d'application des régles de transformation introduites section 4, les formes possibles de préfixe d'une assertion A_t sont considérées. Plus précisement, on abrégera la notation de la forme du préfixe d'une assertion en écrivant la suite ordonnée des quantificateurs qui composent son préfixe. Ainsi, l'assertion ($\forall r_1 \in R_1$) ($\exists r_2 \in R_2$) aura pour forme de préfixe la séquence de caractéres " $\forall \exists$ ". Une notation * sera utilisée pour signaler l'existence d'une ou plusieurs occurences successives d'un même symbole. En particulier la notation Q* désigne une ou plusieurs occurences successives de quantificateurs Q quelconques. Par la suite, pour chaque type de mise à jour, les formes de préfixe d'une assertion A_t conduisant à des simplifications sont étudiées.

4.6.1. Régles en insertion.

Considérons les régles de simplification de A_t dans le cas d'une insertion. Plusieurs classes d'assertion sont simplifiables par l'utilisation de ces régles.

1- D'aprés le théorème 7 les assertions de la forme $\forall^*Q^*$ sont décomposables si il n'y a pas de prédicats de relation quantifiés du type $(\forall r \in R \cup R^+)$ dans la partie de préfixe Q^*. En particulier, dans le cas d'assertions du type $\forall^*\exists^*$, le théorème 8 peut être utilisé aprés la décomposition de chaque prédicat de relation universellement quantifié s'il n'existe pas de prédicats de relation quantifiés du type $(\exists r \in R \cup R^+)$. En effet, il ne reste alors plus de prédicat du type $(\forall r \in R \cup R^+)$; les prédicats de relation du type $(\exists r \in R)$ sont alors remplacés par $(\exists r \in \overline{R})$. La même démarche s'applique aux assertions du type $\forall^*\exists^*Q^*$.

Forme du préfixe	Mises à jour non autorisées	Régles
$\forall^*\exists^*$	..... sans prédicat $(\exists r \in R \cup R^+)$	théorème 7, théorème 8
$\forall^*Q^*$	sans prédicat $(\forall r \in R \cup R^+)$ dans Q^*	théorème 7
$\forall^*\exists^*Q^*$	sans prédicat $(\forall r \in R \cup R^+)$ sans prédicat $(\exists r \in R \cup R^+)$	théorème 7, théorème 8
$\exists^*Q^*$	sans prédicat $(\exists r \in R \cup R^+)$ dans Q^*	théorème 10, théorème 8
$\exists\forall^*Q^*$	sans prédicat $(\forall r \in R \cup R^+)$ dans Q^* sans prédicat $(\exists r \in R \cup R^+)$ dans Q^*	théorème 9, théorème 8

Fig. IV.3 : Assertions simplifiables lors d'insertions

2- Le théorème 9 ajoute à la classe d'assertions traitée ci-dessus les assertions du type $\exists\forall^*\exists^*$.

3- Enfin, les assertions du type $\exists^*Q^*$, dans lesquelles aucune variable dans Q quantifiée par $\exists$ ne porte sur une relation mise à jour, sont simplifiables d'abord par application du théorème 10 puis par application du théorème 8.

Notons que les assertions du type $\forall^*\exists^*$ correspondent à la classe des dépendances [Abiteboul85]. Les cas d'assertions simplifiables sont synthétisés dans le tableau en figure IV.3 ci-dessus.

4.6.2. Régles en suppression

1- Le premier cas de simplification suppose simplement qu'aucune variable quantifiée par $\forall$ portant sur une relation mise à jour, ne soit dominée par $\exists$. Dans ce cas, chaque symbole de relation, R, dans un prédicat de relation quantifié du type $(\exists r \in R)$ ou $(\exists r \in R - R^-)$, peut être remplaçé par $\bar{R}$.

2- Les autres simplifications dépendent étroitement de la forme du préfixe de l'assertion. Un cas particulier se produit si le préfixe est du type $\exists\forall Q^*$ et qu'il n'y a pas de prédicats de relation $(\exists r \in R - R^-)$ dans Q^*. Alors, le théorème 12 s'applique.

3- Considérons le cas des assertions du type $\forall^*\exists^*$. Dans ce cas, le théorème 11 s'applique et la portée, disons R, de chaque variable quantifiée par $\forall$ peut être réduite à R' à l'aide de la formule donnée dans ce théorème. Ce cas s'étend aisément aux assertions de préfixe $\forall^*\exists^*Q^*$ dans lesquelles il n'existe pas de prédicat de relation quantifié $(\exists r \in R - R^-)$ dans Q^*. D'un autre côté, ce type d'assertions de forme de préfixe $\forall^*\exists^*$ satisfait les conditions d'application du théorème 12. De la même manière, ce résultat s'étend aux cas de forme de préfixe du type $\forall^*\exists^*Q^*$ si il n'existe pas de prédicats du type $(\forall r \in R - R^-)$ dans la partie de préfixe Q^*.

4- Considérons finalement le cas des assertions de type $\exists^*Q^*$. Le théorème 12 s'applique s'il n'existe pas de prédicats de relation du type $(\forall r \in R - R^-)$ dans la partie Q^*. Par contre, le théorème 11 ne peut s'appliquer.

Le tableau de la figure IV.4 synthétise ces résultats.

Forme du préfixe	Mises à jour non autorisées	Régles
Q^*	sans prédicat $(\forall r \in R - R^-)$	théorème 12
$\exists\forall Q^*$	sans prédicat $(\exists r \in R - R^-)$ dans Q^*	théorème 12
$\forall^*\exists^*$		théorème 11, théorème 12
$\forall^*\exists^*Q^*$	sans prédicat $(\exists r \in R - R^-)$ dans Q^* sans prédicat $(\forall r \in R - R^-)$ dans Q^*	théorème 11, théorème 12
$\exists^*Q^*$	sans prédicat $(\forall r \in R - R^-)$ dans Q^*	théorème 12

Fig. IV.4 : Assertions simplifiables lors de suppressions

Il apparait clairement que les types d'assertions de préfixe $\forall^*\exists^*$, $\exists\forall^*\exists^*$, $\forall^*\exists^*Q^*$, $\exists^*Q^*$ ou $\exists\forall^*\exists^*Q^*$ sont simplifiables avec plus ou moins de restrictions sur les types de mises à jour. L'alternance de quantificateurs dans le préfixe rend plus difficile, voire souvent *impossible* l'application des régles de simplification bien qu'un pré-test différentiel puisse être obtenu.

5. TRAITEMENT D'UNE DEFINITION D'ASSERTION

5.1. Introduction

Dans cette section, seul le traitement de la définition d'une contrainte "générale" est abordé puisque celle-ci correspond à une assertion relationnelle. Certaines limitations sont imposées sur la forme d'une assertion. Celles-ci se réduisent, dans la première version implantée du module COMPILATEUR du sous système d'intégrité de Sabrina [Borla85], à des assertions mono-variable multi-relations ; c'est-à-dire à des assertions où une seule variable tuple peut être définie sur une même relation. Cette restriction n'ampute pas la généralité des observations effectuées sur le comportement de la méthode de simplification et le gain apporté. De plus, la modularité du COMPILATEUR, implanté comme un automate d'états finis, autorise aisément l'enrichissement de la méthode par le rajout de règles de simplification dans le cas d'assertions multi-variable multi-relations. Le principe des algorithmes de traitement d'une assertion est le suivant. Etant donné une assertion définie par l'usager, il s'agit, pour chaque prédicat de relation quantifié du préfixe, de déterminer toutes les simplifications applicables. Pour celà, on considère successivement chaque type de mise à jour, puis les règles de simplification vues dans la Section 4 sont appliquées. En particulier, si chaque relation apparaît dans au plus un prédicat de relation quantifié, deux formes simplifiées au plus sont associées à chaque relation. En effet, d'après le théorème 3, le type du quantificateur déterminera s'il existe une contrainte compilée en insertion ou en suppression. Une contrainte compilée en modification ne sera obtenue que si la contrainte compilée en suppression existe. Ce dernier point sera détaillé par la suite.

Cette section s'organise donc comme suit. Pour chaque type de mise à jour, l'algorithme général permettant de générer des pré-tests différentiels simplifiés ou non est donné. Cet algorithme s'applique à tout type d'assertion. Puis, l'algorithme particulier correspondant aux cas d'assertions mono-variable est décrit, accompagné d'un exemple illustratif. Ensuite, la méthode de passage à des contraintes compiles est présentée. Elle consiste à transformer chaque pré-test différentiel en une sous-question exécutable par le SGBD.

5.2. Algorithmes de génération des pré-tests différentiels

5.2.1. Cas d'une insertion

Les notations utilisées sont identiques à celles introduites dans la Section 4 précédente. La mise à jour t est t (R) := $R \cup R^+$. Dans ce cas, deux types de simplification sont possibles. La première correspond aux théorèmes 7 et 9. Elle s'applique aux variables r quantifiées universellement et non précédées par un quantificateur existentiel sauf peut être le premier quantificateur du préfixe. Le second type de simplification est donné par l'application du théorème 8 aux quantificateurs existentiels de l'assertion qui précèdent le prédicat de relation contenant la relation mise à jour. L'algorithme général applicable à tout type d'assertion peut maintenant être énoncé.

Notation :

L'expression E $\leftarrow$ A signifie que E est remplacé par A (ou E est affecté à A).

Algorithme général d'insertion :

L'algorithme suit les étapes suivantes :

Etape 1 : prendre tous les prédicats de relation quantifiés de A_t du type ($\forall r_i \in R \cup R^+$). Puis appliquer successivement en commençant par la fin de l'assertion la règle de simplification donnée par les théorèmes 7 et 9 à chaque expression ($\forall r_i \in R \cup R^+$) F. Si la règle ne s'applique pas alors remplacer toute sous formule du type ($\forall r_i \in R \cup R^+$) F par ($\forall r_i \in R$) F $\wedge$ ($\forall r_i \in R^+$) F en utilisant le théorème 4 puis utiliser la règle de décomposition donnée par le théorème 5.

Proposition 1 : Le résultat de l'étape 1 est une expression équivalente à A_t de la forme :

A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall, r_i : R] \wedge G$

où G est une formule (i.e., pas encore un pré-test différentiel).

Preuve :

La preuve est élémentaire. En effet, posons f = ($\forall r_i \in R$) F $\wedge$ ($\forall r_i \in R^+$) F. Si f est dominée par ($\exists r_{i-1} \in R_{i-1}$) alors on obtient d'après le théorème 5 que :
($\exists r_{i-1} \in R_{i-1}$) f $\Leftrightarrow$ ($\exists r_{i-1} \in R_{i-1}$) ($\forall r_i \in R$) F $\wedge f_1$. Si maintenant f est dominée par ($\forall r_{i-1} \in R_{i-1}$), alors on obtient que ($\forall r_{i-1} \in R_{i-1}$) f $\Leftrightarrow$ ($\forall r_{i-1} \in R_{i-1}$) ($\forall r_i \in R$) F $\wedge$ ($\forall r_{i-1} \in R_{i-1}$) ($\forall r_i \in R^+$) F. L'application de la règle de simplification est un cas particulier du théorème 5. Donc au total, en itérant sur tous les prédicats de relation quantifiés précédant f puis sur chaque i tel que $R_i = R$, on obtient la proposition. G est alors une formule composée de la conjonction de toutes les sous formules générées à chaque itération à l'exception de la formule A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall, r_i : R]$. ♦

Etape 2 : prendre tous les prédicats de relation quantifiés du type ($\exists r_j \in R \cup R^+$) dans chacune des assertions A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall, r_i : R]$ et G. Appliquer successivement en partant de la fin de l'assertion la règle de simplification du théorème 8 et la règle de décomposition du théorème 10 à toute sous formule du type ($\exists r_j \in R \cup R^+$) F.

Proposition 2 : Le résultat de l'étape 2 est une expression équivalente à A_t de la forme : G' où G' est l'expression simplifiée (i.e., le pré-test différentiel) résultant de G.

Preuve :

Posons $f = (\exists r_j \in R \cup R^+)\ F$ une sous formule de A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall$, $r_i : R$] telle que F ne contienne aucun prédicat du type $(\exists r_j \in R \cup R^+)$. Par application du théorème 4 on a $f = (\exists r_j \in R)\ F \vee (\exists r_j \in R^+)\ F$. Si f est dominée par $(\exists r_{i-1} \in R_{i-1})$ alors on obtient d'après le théorème 5 : $f = (\exists r_{i-1} \in R_{i-1})(\exists r_j \in R)\ F \vee (\exists r_{i-1} \in R_{i-1})(\exists r_j \in R^+)\ F$. Si f est dominée par $(\forall r_{i-1} \in R_{i-1})$ alors on a : $f = (\forall r_{i-1} \in R_{i-1})\ (\exists r_j \in R)\ F \vee f_1$. En itérant sur tous les prédicat de relation quantifiés précédant f et sur tous les j on obtient une expression, équivalente à A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall, r_i : R$], qui est une disjonction d'assertions (pré-test différentiels). Il suffit alors de remarquer qu'un de ces pré-tests est précisement :
A_t [pour tout i tel que $R_i = R$, $r_i : R$] c'est à dire A. Par suite, puisque ce terme est vrai il rend vrai tout le terme conjonctif équivalent à A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall, r_i : R$]. D'où le résultat. ♦

Etape 3 : l'expression G' constitue le pré-test différentiel dérivé de l'assertion A. De plus, si les règles de simplification se sont appliquées on a obtenu une contrainte simplifiée.

{fin de l'algorithme général}

Un commentaire s'impose sur l'algorithme général présenté ci-dessus. Cet algorithme ne génère pas dans tous les cas un pré-test différentiel simplifié. Lorsque les règles de simplification s'appliquent, une forme simplifiée est obtenue. Dans le cas contraire, l'assertion est juste décomposée en un pré-test différentiel qui permet de contrôler l'assertion sur l'état de la base avant mise à jour.

Nous décrivons dans ce qui suit l'algorithme implanté dans le cas des assertions multi-relation mono-variable.

Algorithme 1 :

Soit A une assertion de la forme :
(Q1 r1 ∈ R1) (Q2 r2 ∈ R2) ... (Qn rn ∈ Rn) (M).
Soit Qj = ∀. L'assertion compilée associée à $\Re j$ est ($\Re j$, insérer, E) où E est obtenu par application des transformations successives :

E ← A

Si (il existe au plus un quantificateur existentiel Qi dominant Qj et qu'il n'existe pas de quantificateur universel dominant Qi) ou (aucun quantificateur existentiel domine Qj)

alors

$E \leftarrow E\,[ri : Ri]$

$E \leftarrow E\,[rj^+ / rj,\ rj^+ : Rj^+]$

sinon

Soient $Q_{e1}, \ldots, Q_{ek}$ les quantificateurs existentiels dominant Qj

$E \leftarrow E\,[r_{ek} : \overline{R}_{ek}]$

(*) $E \leftarrow E\,[(\forall r_j^+ \in R_j^+)\ M\,[r_j^+ / r_j]\ ET\ M / M]$. ♦

Notons que la dernière transformation (*) correspond à l'expression sous forme de pré-test différentiel de l'assertion résultat de la transformation :

$E \leftarrow E\,[rj : Rj \cup Rj^+]$

Exemple 12 :

Soit la contrainte référentielle :

$A = (\forall a \in ABUS)\ (\exists v \in VINS)\ (a.nv = v.nv)$.

L'assertion compilée associée à ABUS est (ABUS, insérer, E) où E est obtenue par :

$E := A$

$E := (\forall a^+ \in ABUS^+)\ (\exists v \in VINS)\ (a^+.nv = v.nv)$. ♦

Soit maintenant l'assertion :

$A = (\exists v \in VINS)\ (\exists b \in BUVEURS)\ (\forall a \in ABUS)\ (a^+.nv \neq v.nv$ et $a.nb \neq b.nb)$.

L'assertion compilée associée à ABUS est (ABUS, insérer, E) où E est obtenue par :

$E := A$

$E := (\exists v \in \overline{VINS})\ (\exists b \in \overline{BUVEURS})\ (\forall a \in ABUS)\ (\forall a^+ \in ABUS^+)$.

$(a^+.nv \neq v.nv \wedge a^+.nb \neq b.nb) \wedge (a.nv \neq v.nv \wedge a.nb \neq b.nb)$. ♦

Par suite, le cas de cette dernière assertion montre qu'une simplification ne peut être obtenue que si les ensembles $\overline{VINS}$ et $\overline{BUVEURS}$ sont maintenus de manière redondante dans la base, par exemple comme des vues concrètes [Blaustein81, Blakeley86].

Les expressions respectives du calcul de VINS et BUVEURS sont :

$\overline{VINS} = \{\ t \in VINS / A\,[v : t\,]\}$ et

$\overline{BUVEURS} = \{\ t \in BUVEURS / A\,[b : t]\}$.

Dans l'hypothèse où de tels ensembles sont efficacement maintenus à jour lors des modifications de la base, il est possible d'accepter des assertions mono-variables multi-relations du type :

(1) $\exists\exists \ldots \exists\ \forall\forall \ldots \forall\ (M)$

Dans le cas contraire, le recalcul des ensembles $\overline{VINS}$, $\overline{BUVEURS}$ rend l'évaluation de l'assertion modifiée, E, beaucoup plus chère que l'évaluation de l'assertion A originale. Donc, les assertions du type (1) ci-dessus seraient rejetées par le compilateur.

5.2.2. Cas d'une suppression

La mise à jour est du type t (R) := R - R^-. La première simplification est donnée par le théorème 12. Elle consiste à remplacer, pour chaque prédicat de relation ($\exists r_i \in R_i$) quantifié existentiellement et non précédée par un quantificateur universel, le nom de relation Ri par l'ensemble redondant $\overline{Ri}$ associé. Le second type de simplification concerne les prédicats de relation universellement quantifiés précédant le prédicat de relation contenant la relation mise à jour. Le théorème 11 peut s'appliquer dans les conditions d'application décrites Section 4. L'algorithme général suivant peut donc être énoncé.

Algorithme général

L'algorithme suit les étapes suivantes.

Etape 1 : prendre tous les prédicats de relation quantifiés de A_t du type ($\exists r_i \in R - R^-$). Appliquer successivement en commençant par la fin de l'assertion les règles de simplification données par les théorèmes 11 et 12. Puis remplacer toute sous formule du type ($\exists r_i \in R - R^-$) F par ($\exists r_i \in R$) F $\wedge$ A' en utilisant le théorème 4 puis la règle de décomposition donnée par le théorème 5.

Proposition 3 : Le résultat de l'étape 1 est une expression équivalente à A_t de la forme :

A_t [pour tout i t.q $R_i = R$ et $Q_i = \exists, r_i : R] \wedge G$

où G est une formule.

Preuve :

La preuve est élémentaire. En effet, posons f = ($\exists r_i \in R - R^-$) F. f s'écrit aussi d'après le théorème 4 : f = ($\exists r_i \in R$) (F $\wedge \neg R^- (r_i)$). Ce qui s'écrit aussi d'après le théorème 5 : f = ($\exists r_i \in R$) F $\wedge$ A'. Si f est dominée par ($\exists r_{i-1} \in R_{i-1}$) alors on obtient d'après le théorème 5 que ($\exists r_{i-1} \in R_{i-1}$) f $\Leftrightarrow$ ($\exists r_{i-1} \in R_{i-1}$) ($\exists r_i \in R$) F $\wedge f_1$. Si maintenant f est dominée par ($\forall r_{i-1} \in R_{i-1}$), alors on obtient que ($\forall r_{i-1} \in R_{i-1}$) f $\Leftrightarrow$ ($\forall r_{i-1} \in R_{i-1}$) ($\exists r_i \in R$) F $\wedge$ ($\forall r_{i-1} \in R_{i-1}$) A'. L'application de la règle de simplification donnée par le théorème 11 est un cas particulier de l'application du théorème 5 aux prédicats de relation quantifiés universellement qui précèdent f dans A_t. Quant à la simplification donnée par le théorème 12, elle ne modifie pas la forme de l'assertion. Donc au total, en itérant sur tous les prédicats de relation quantifiés précédant f puis sur chaque i tel que $R_i = R$, on obtient la proposition. ♦

Etape 2 : prendre tous les prédicats de relation quantifiés du type ($\forall r_j \in R - R^-$) dans chacune des assertions A_t [pour tout i t.q $R_i = R$ et $Q_i = \exists, r_i : R]$ et G. Appliquer successivement en partant de la fin de l'assertion la règle de décomposition du théorème 4 puis du théorème 6 à toute sous formule du type ($\forall r_j \in R - R^-$) F.

Proposition 4 : Le résultat de l'étape 2 est une expression équivalente à A_t de la forme : G' où G'

est l'expression simplifiée (i.e., le pré-test différentiel) résultant de G.

Preuve :

Posons $f = (\forall r_j \in R - R^-)$ F une sous formule de A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall$, $r_i : R$] telle que F ne contienne aucun prédicat du type $(\forall r_j \in R - R^-)$. Par application du théorème 4 on a $f = (\forall r_j \in R)(F \vee R^-(r_i))$. Ce qui s'écrit encore d'après le théorème 5, $f = (\forall r_j \in R)\ F \vee A'$. Si f est dominée par $(\exists r_{i-1} \in R_{i-1})$ alors on obtient d'après le théorème 5 :
$f = (\exists r_{i-1} \in R_{i-1})(\forall r_j \in R)\ F \vee (\exists r_{i-1} \in R_{i-1})\ A'$. Si f est dominée par $(\forall r_{i-1} \in R_{i-1})$ alors on a $f = (\forall r_{i-1} \in R_{i-1})\ (\exists r_j \in R)\ F \vee f_1$. En itérant sur tous les prédicat de relation quantifiés précédant f et sur tous les j on obtient une expression, équivalente à A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall, r_i :$ R], qui est une disjonction d'assertions (pré-test différentiels). Il suffit alors de remarquer qu'un de ces pré-tests est précisement :
A_t [pour tout i tel que $R_i = R$, $r_i : R$] c'est à dire A. Par suite, puisque ce terme est vrai il rend vrai tout le terme conjonctif équivalent à A_t [pour tout i t.q $R_i = R$ et $Q_i = \forall, r_i : R$]. D'où le résultat. ♦

Etape 3 : l'expression G' constitue le pré-test différentiel dérivé de l'assertion A. De plus, si les règles de simplification se sont appliquées on a obtenu une contrainte simplifiée.

{fin de l'algorithme général}

Nous présentons maintenant l'algorithme particulier implanté dans le cas d'assertions mono-variables.

Algorithme 2 :

Soit A une assertion de la forme :

$A = (Q1\ r1 \in R1) \ldots (Qn\ rn \in Rn)\ (M)$

Soit $Qj = \exists$.

Soit $Qk = \forall$ avec $k < j$. L'assertion compilée en suppression associée à $\Re j$ est ($\Re j$, supprimer, E) où E est obtenu par application des transformations successives :

$E \leftarrow A$

Si (il existe au plus un quantificateur existentiel Qi dominant Qk et qu'il n'existe pas de quantificateur universel dominant Qi) ou (aucun quantificateur existentiel ne domine Qk)

et si le préfixe compris entre Qk et Qj est de la forme : $\forall^* \exists^*$

alors

$E \leftarrow E\ [\bar{R}j / Rj]$

(1) $E \leftarrow E\ [((\forall r^-j \in R^-j)\ (rj \neq r^-j \wedge M) / M]$

(2) $E \leftarrow E\ [(As\ [r'^-j / sj, r'^-j : R^-j] \Rightarrow M / M]$

sinon si il n'existe pas de quantificateur Qk précédant Qj

alors

(3) $E \leftarrow E\ [((\forall\ r^-j \in R^-j)\ (r^-j \neq rj) \wedge M) / M]$

Soient Qe1, ... , Qek les quantificateurs existentiels non dominés par un quantificateur universel alors

$E \leftarrow E\ [rek : Rek]$. ♦

Remarques :

Les transformations (1) et (2) correspondent simplement à la transformation

$E \leftarrow E\ [\ R'k / Rk\ ]$

avec $R'k = \{\ t \in Rk / A\ [rk : t, r^-j / rj, r^-j : R^-j]\}$.

(1) et (2) permettent l'obtention d'un pré-test différentiel :

(1) exprime que dans E, rj prend ses valeurs dans $Rj-R^-j$

(2) exprime que seulement pour les tuples rk de R'k, la matrice M de l'assertion doit être évaluée. On a donc : As = A où toutes les variables r_i sont remplacées par des variables s_i (ce renommage permet d'éviter les conflits entre variables dans E), et où le prédicat de relation quantifié ($\forall\ r_k \in R_k$) est ôté du préfixe de A.

Finalement, la transformation (3) joue le même rôle que (1).

Exemple 13 :

Soit la contrainte référentielle :

$A = (\forall\ a \in ABUS)\ (\exists\ v \in VINS)\ (a.nv = v.nv)$

et la mise à jour t (VINS) := VINS - $VINS^-$.

L'assertion compilée associée à VINS, en suppression, est (VINS, supprimer, E) où E est obtenue par :

$E := A$

$E := (\forall\ a \in ABUS)\ (\exists\ v \in VINS)\ (\forall\ v^- \in VINS^-)\ ((av.nv \neq v^-.nv) \wedge (a.nv = v.nv))$

$As = (\exists\ v' \in VINS)\ (a.nv = v'.nv)$ et donc,

$As\ [\ v'^- / v'] = (\exists\ v'^- \in VINS^-)\ (a.nv = v'^-.nv)$

Donc :

$E := (\forall\ a \in ABUS)\ (\exists\ v \in VINS)\ (\forall\ v^- \in VINS^-)\ ((\exists\ v'^- \in VINS^-)$

$(a.nv = v'^-.nv)) \Rightarrow ((v.nv \neq v^-.nv) \wedge (a.nv = v.nv))$

Dans ce cas particulier, une simplification supplémentaire peut être effectuée, puisque nv est clé de la relation VINS. En effet, si $(\exists\ v'^- \in VINS^-)\ (a.nv = v'^-.nv)$ est VRAI alors il n'est pas possible de trouver v tel que $(v.nv \neq v^-.nv)$ et $(a.nv = v.nv)$. Donc, si le premier membre de l'implication est vrai, alors le second membre est FAUX et E devient FAUX. D'un autre côté, si le premier membre est FAUX, alors E est VRAI. Par suite, E peut s'écrire :

$E := (\forall\ a \in ABUS)\ (\forall\ v^- \in VINS^-)\ (a.nv \neq v^-.nv)$. ♦

Un autre algorithme particulier peut être utilisé en application du théorème 12 dans le cas particulier où la suppression porte sur la première relation du préfixe. Dans ce cas, la contrainte devient parfaitement simplifiable. La contrainte compilée ($\Re_j$, supprimer, E) s'obtient comme suit.

Algorithme 3 :

Soit Qi = ∃ et t (Ri) := Ri-Ri⁻

si i = 1

alors

E ← (∃ r1 ∈ $\overline{R1}$) (∀ r⁻1 ∈ R⁻1) (r1. ≠ r⁻1). ♦

5.2.3. Cas d'une modification

Une modification d'une relation R est traitée comme une suppression de tuples de R suivie par leur insertion, aprés modification, dans R. Ceci est équivalent à supprimer des tuples sachant qu'ils seront réinsérés avec de nouvelles valeurs. Par conséquent, il n'est pas possible de contrôler les contraintes compilées en suppression sur R puis les contraintes compilées en insertion sur R puisque ces deux opérations ne sont pas indépendantes. En fait, il est nécessaire d'introduire un type particulier de contrainte compilée en modification. L'algorithme suivant génère d'abord dans le cas général de telles contraintes. Cet algorithme travaille avec l'hypothèse suivante. On suppose que lors d'une modification d'une relation R, t (R) := (R - R⁻) ∪ R⁺, R⁻ ∩ R⁺ = ∅. Dés lors, t (R) est équivalent à (R ∪ R⁺) - R⁻; on peut donc traiter dans un ordre quelconque les deux mises à jour insertion et suppression. C'est le principe utilisé par l'algorithme qui suit.

Algorithme général

L'algorithme suit les étapes suivantes.

Etape 1 : prendre tous les prédicats de relation quantifiés de A_t du type (∃ r_i ∈ (R ∪ R⁺) - R⁻) et du type (∀ r_j ∈ (R - R⁻) ∪ R⁺). Appliquer successivement en commençant par la fin de l'assertion les règles de décomposition données par les théorèmes 4 et 6 à toutes les sous formules du type (∃ r_i ∈ (R ∪ R⁺) - R⁻) F et (∀ r_i ∈ (R - R⁻) ∪ R⁺) F.

Proposition 5 : Le résultat de l'étape 1 est une expression équivalente à A_t de la forme :

A_t [pour tout i t.q R_i = R et Q_i = ∃, r_i : R ∪ R⁺; pour tt j tq R_j = R et Q_j = ∀, r_j : R - R⁻] ∧ C

où G est une formule.

Preuve :

La preuve découle immédiatement des propositions 1 et 3. ♦

Etape 2 : prendre tous les prédicats de relation quantifiés du type (∀ r_j ∈ R - R⁻) et du type

$(\exists r_j \in R \cup R^+)$ dans chacune des deux formules obtenues à l'étape 1. Appliquer successivement en partant de la fin de l'assertion les règles de décomposition du théorème 4 puis du théorème 5 à toute sous formule du type $(\forall r_j \in R - R^-) F$ et $(\exists r_j \in R \cup R^+) F$.

Proposition 6 : Le résultat de l'étape 2 est une expression équivalente à A_t de la forme : G' où G' est l'expression simplifiée (i.e., le pré-test différentiel) résultant de G.

Preuve :

La preuve résulte directement des propositions 2 et 4. En effet, au terme de l'étape 2, la formule A_t [pour tout i t.q $R_i = R$ et $Q_i = \exists$, $r_i : R \cup R^+$; pour tt j tq $R_j = R$ et $Q_j = \forall$, $r_j : R - R^-$] sera transformée en une disjonction de formules dont l'une d'entre elles sera précisément l'assertion A. D'où le résultat. ♦

Etape 3 : l'expression G' constitue le pré-test différentiel dérivé de l'assertion A. De plus, si les règles de simplification se sont appliquées on a obtenu une contrainte simplifiée.

{fin de l'algorithme général}

Remarque : dans l'algorithme ci-dessus les règles de simplification s'appliquent dans les formules où tous les prédicats de relation quantifiés évoluent de la même manière (i.e., soit en insertion soit en suppression). Ce cas se produit par exemple lorsqu'un seul prédicat de relation quantifié est mis à jour ou bien lorsque tous les prédicats de relation sont quantifiés de la même manière. L'algorithme particulier qui suit illustre une de ces possibilités (cas d'assertion mono-variable).

Algorithme 4 :

Soit $Q_j = \exists$ dans une assertion A.

La contrainte compilée associée à $\Re_j$ en modication est ($\Re_j'$, modification, E') où E' est obtenue par les transformations successives suivantes :

Soit ($\Re_j$, supprimer, E) la contrainte compilée associée à $\Re_j$ en supression,

si il n'existe pas de quantificateur universel dominant Q_j

alors $E' \leftarrow E \vee A\ [r_j^+ / r_j, r_j^+ : R_j^+]$

sinon

soit $Q_k = \forall$ et $k < j$,

si (il existe au plus un quantificateur existentiel Q_i dominant Q_k et qu'il n'existe pas de quantificateur universel dominant Q_i) ou (aucun quantificateur existentiel ne domine Q_k) et si le préfixe compris entre Q_k et Q_j est de la forme : $\forall^* \exists^*$

alors

(1) $E' \leftarrow E'\ [((\forall r_j^- \in R_j^-)(r_j \neq r_j^-)$ ET M) $\vee$ ($(\exists r_j^+ \in R_j^+)$ M $[r_j^+/r_j]$)) | M]

(2) $E' \leftarrow E'\ [R_k'/R_k]$

sinon

$$E' \leftarrow E' [((\forall r_j^- \in R_j^-)(r_j \neq r_j^-) \text{ ET } M) \vee ((\exists r_j^+ \in R_j^+) M [r_j^+/r_j])) \mid M] \ \blacklozenge$$

Remarques :

La transformation (1) permet simplement d'exprimer sous forme de pré-test différentiel l'assertion A_t contenant le prédicat de relation quantifiée $(\exists r_j \in (R_j - R_j^-) \cup R_j^+)$.

La transformation (2) est traduite sous forme de pré-test différentiel de la même manière que dans l'algorithme 2.

Exemple 14 :

Soit l'assertion A :

$$A = (\forall a \in ABUS) (\exists v \in VINS) (a.nv = v.nv)$$

et la mise à jour t (VINS) := (VINS-VINS$^-$) $\cup$ VINS$^+$.

La contrainte compilée associée à VINS, en modification, est (VINS.nv, modifier, E') où E' est obtenue par :

$$E' := (\forall a \in ABUS) (\exists v \in VINS) ((\forall v_1^- \in VINS^-) (v_1^- \neq v) \wedge (a.nv = v.nv)) \vee (\exists v^+ \in VINS^+) (a.nv = v^+.nv))$$

$$E' := (\forall a \in ABUS) (\exists v \in VINS)[(\exists v_2^- \in VINS^-) (a.nv = v_2^-.nv)] \Rightarrow [((\forall v_1^- \in VINS^-) (v_1^- \neq v) \wedge (a.nv = v.nv)) \vee (\exists v^+ \in VINS^+) (a.nv = v^+.nv)]$$

Cette expression de E' peut être encore simplifiée en remarquant que nv est clé de la relation VINS. Par suite, en utilisant un raisonnement similaire à l'exemple 2, le terme $(\forall v_1^- \in VINS^-) (v_1 \neq v) \wedge$ (a.nv = v.nv) est inutile. Il prend la valeur FAUX quand le premier membre de l'implication est VRAI . Donc E' se réduit à :

$$E' := (\forall a \in ABUS)[(\forall v_2^- \in VINS^-) (a.nv \neq v_2^-.nv)] \vee (\exists v^+ \in VINS^+)(a.nv = v^+.nv)]$$

Ce qui se simplifie encore en :

$$E' := (\forall a \in ABUS)[(\forall v_2^- \in VINS^-) (a.nv \neq v_2^-.nv)] \vee (\exists v^+ \in VINS^+)(v_2^-.nv = v^+.nv)] \ \blacklozenge$$

5.3. Génération des contraintes compilées

Une contrainte compilée est un triplet associant, à un schéma relationnel $\Re$ et un type de mise à jour T, un pré-test différentiel E à vérifier. Le propos de cette section est de transformer chaque pré-test différentiel en une requête relationnelle permettant de retrouver les tuples qui sont mis à jour et qui satisfont le pré-test différentiel .

Plusieurs questions se posent :

(1) Est-il toujours possible de trouver une requête relationnelle exécutable correspondant à un pré-test différentiel ?

(2) Dans le cas où une requête relationnelle exécutable est construite, le langage permet-il de capturer toutes les optimisations inhérentes à l'évaluation du pré-test différentiel.

Ces questions sont dicutées en détail dans [Simon86].

6. CONCLUSION

Dans ce papier, nous avons présenté une méthode originale de simplification d'assertions basée sur la notion de pré-test différentiel et de contrainte compilée. Cette méthode s'appuie sur l'heuristique simple qui consiste à réduire la taille du produit cartésien des relations référençées dans une assertion. Pour cela, nous supposons que les mises à jour sont trés sélectives c'est à dire que le volume de données mis à jour par une instruction est petit par rapport à la taille de la relation mise à jour. Sous cette hypothése, nous avons tout d'abord défini des régles de décomposition d'une assertion en un pré-test différentiel en fonction du type de la mise à jour (insérer ou supprimer). Le problème reste qu'il est possible de générer des pré-tests différentiels plus coûteux à évluer que l'assertion originale. Nous proposons alors des règles de simplification qui permettent de déterminer les classes d'assertions pour lesquelles les pré-tests différentiels générés sont "en général" plus simples à évaluer que l'assertion originale. De plus, les règles de simplification introduisent des optimisations supplémentaires via la maintenance d'informations redondantes. La méthode proposée possède ainsi les avantages suivants:

(1) Son activation s'effectue une fois pour toutes au moment de la définition d'une contrainte et permet un stockage efficace des contraintes sous une forme compilée.

(2) La méthode permet un contrôle préventif des incohérences dans la base de données ce qui évite, en cas d'incohérence, à la fois d'avoir à défaire une mise à jour (opération qui peut être minimisée dans certains systémes) mais aussi de perdre le temps consacré à la construction des chemins d'accés pour les tuples ajoutés à la base.

(3) La méthode est générale. Elle supporte une large classe d'assertions multi-variable multi-relations et accepte la mise à jour ensembliste d'une relation.

(4) Pour chaque assertion A, la méthode détecte les mises à jour qui préservent trivialement A. Pour les autres types de mise à jour, A est transformée en un ensemble de contraintes compilées de la forme (R, T, E) où R est une relation de A, T est un type d'opération de mise à jour (insérer, supprimer ou modifier) et E est une précondition optimisée que doivent vérifier les tuples de R mis à jour par T de manière à garantir la satisfaction de A. Par pré-condition optimisée, on entend que le volume de données à accéder pour vérifier E est beaucoup plus faible que celui qui est nécéssaire pour vérifier A.

Le bilan des régles de simplification que nous avons définies nous a conduit à isoler une sous classe d'assertions simplifiables. Cette sous classe recouvre la classe des dépendances étudiée par exemple dans [Abiteboul85].

Une première version de la méthode de simplification décrite ici a été implantée dans le sous système d'intégrité de SABRINA pour des assertions multi-relation mono-variable. Pour chaque type de mise à jour, nous avons décrit les algorithmes permettant de construire des pré-tests différentiels à partir d'une assertion. Ces algorithmes ont été implantés au moyen d'un automate d'états fini décrit dans [Borla85].

La réalisation d'un premier prototype de notre méthode a permis d'envisager de nouvelles améliorations pour accroitre les performances du contrôle d'intégrité. Ces améliorations sont les suivantes. La première amélioration consiste à prendre en compte la matrice de l'assertion dans le processus de simplification. Une façon de faire est de mettre les assertions sous une forme non prénexe où la portée des quantificateurs est réduite à une sous formule minimale. Ceci entraine d'abord une complication des règles de décomposition et de simplification puisque tous les cas de construction d'une formule (différentes connectives) doivent être considérés. De plus, cela complique l'implantation de la méthode et nécessite des structures de données adaptés à la manipulation de ces formules. Une telle amélioration est étudiée dans [Borla87]. La deuxième amélioration tient dans la prise en compte de la nature de la mise à jour. En particulier, si l'on connait exactement les tuples modifiés d'après l'expression du calcul relationnel qui les qualifie, il est possible de simplifier plus encore l'assertion (par ex. en détectant des sous formules contradictoires, redondantes ou partiellement redondantes). Enfin, la dernière amélioration porte sur la simplification possible de l'assertion simplifiée par la prise en compte de l'assertion originale supposée vraie sur l'état non modifié de la base. Ceci peut conduire à détecter des sous formules identiques qui, dans certains cas, n'ont alors plus besoin d'être évaluées.

Remerciements : Je tiens à remercier Serge Abiteboul et Patrick Valduriez pour m'avoir toujours encouragé et aidé à aller au bout de ce travail. Merci aussi à Georges Gardarin pour les nombreuses discussions, toujours très animées et fructueuses, que nous avons eues ensemble. Je suis très reconnaissant à Pascale Borla pour avoir implanté une partie des algorithmes décrits ici. Je remercie enfin François Bancilhon pour ses commentaires sur une version préliminaire de ce travail.

REFERENCES :

Abiteboul85 S. Abiteboul, "*Coktail de dépendances*", Thése de doctorat d'état, Univ. Paris Sud Orsay, Août 1985.

ANSC83 ANSC X3H2 "*SQL Database Language*" ANSI Reports, September 1982 and Feb.1983, New-York, ANSI, 1983.

Astrahan76 M. M. Astrahan & al. : "*System R : Relational approach to database management*", Publ. ACM-TODS Vol. 1, June 1976.

Blaustein81 B. Blaustein : "*Enforcing database assertions.Techniques and applications*", Ph.D Harvard University, August 1981.

Blakeley86 J. A. Blakeley, P. A. Larson, F. Tompa : "*Efficiently updating materialized views*", Proc. of the ACM-SIGMOD Int. Conf., Washington, May 1986.

Borla85 P. Borla, "*Compilation de contraintes d'intégrité exprimées dans un langage de haut niveau*", Mémoire d'ingénieur IIE-CNAM, 1985.

Borla87 P. Borla, E. Simon : "*Simplification methods for integrity control revisited*", en préparation.

Brodie78 M. Brodie : "*Specification and verification of database semantic integrity*", Ph.D Toronto University,March 1978.

Bouzeghoub85 M. Bouzeghoub, G. Gardarin, E. Metais : "*Database design approach : An Expert System Approach*", Proc. 11th VLDB Conf., Stockolhm, August 85.

Buneman79 O.P Buneman, E.K Clemons : "*Efficiently monitoring Relational Databases*", Publ. ACM-TODS 4:3, Sept 1979.

Chamberlin81 D. Chamberlin et al. : "*Support for repetitive transactions and ad-hoc queries in System-R*", ACM TODS, v. 6, N°. 1, March 1981.

Codasyl73 Codasyl Data Description Language Committee, Journal of Development, 1973.

Codd71 E. F. Codd, "A data base sublanguage founded on the relational calculus", Proc. ACM-SIGFIDET'71, pp. 35-68, 1971.

Codd72 E. F. Codd, "Relational Completeness of Data Base Sublanguage" in R. Rustin (ed.), Data base management courant comput. Sci. Symp.6, Prentice Hall, Englewood Cliffs, pp. 65-98, 1972.

Decker86 M. Decker, "Integrity enforcement on Prolog-based deductive databases" in Proc. 1st. Int. Conf. on Exp. Database Syst., Charleston, April 1986.

Eswaran75 K.P. Eswaran, D.D. Chamberlin : "Functional Specifications of a subsystem for database integrity", Proc. VLDB Conf., Framington, Mass., Sept 75.

Fagin84 R. Fagin, M. Vardi, "The theory of data dependencies : a survey", IBM Res. Rep. RJ 4321, Juin 1984.

Findler79 N. Findler : "Associative Networks : Representation and use of knowledge by computer", Academic Press, N.Y, 1979.

Florentin74 J.J. Florentin : "Consistency auditing of databases", The Computer Journal 17, 1974.

Gardarin79 G. Gardarin, M. Melkanoff : "Proving consistency of data base transactions", Proc.5th VLDB Conf.,Tokyo Sept.1979.

Hammer75 H. Hammer, D.J Mc Leod : "Semantic integrity in a relational database system", Proc.1st VLDB, Framingham 1975.

Hammer78 M. Hammer, S.K. Sarin, "Efficient monitoring of data base assertions", Proc. ACM-SIGMOD Conf., Austin June 1978.

Henschen84 L.J Henschen, W.W McCune, S.A Naqvi , "Compiling constraint checking programs from first order formulas", Publ. in Advances in Data Base theory (vol 2), Plenum Press New York, edited by H. Gallaire and J. Minker

Mylopoulos80 J. Mylopoulos, P.A. Bernstein, H.K.T. Wong : "A language facility for designing database-intensive applications", Publ. ACM TODS Vol 5:2, June 80.

Nicolas82 J. M. Nicolas, R. Demolombe : "On the stability of relational queries", Proc. of Int. Workshop : Logical bases for Databases, CERT, Toulouse, 1982.

Nicolas82b J.M Nicolas : "Logic for improving Integrity checking in relational data bases", Acta Informatica, July 1982.

Sheard86 T. Sheard, D. Stemple : "Automatic verification of database transaction safety", Coins Tech. Report 86-30, Univ. Massachussetts at Amherst, 1986.

Simon84 E. Simon, P. Valduriez : "*Design and implementation of an extendible integrity subsystem*", Proc. of ACM-SIGMOD Int. Conf., Boston, June 1984. Paru aussi dans SIGMOD RECORDS Vol. 14-2.

Simon86 E. Simon : "Conception, analyse et réalisation d'un sous système d'intégrité relationnel", Thèse de Doctorat, Univ. de Paris VI, Juin 1986.

Simon87 E. Simon, P.Valduriez : "*Design and analysis of a relational integrity subsystem*", MCC Research Report DB-015-87, Jan. 1987 .

Stonebraker75 M. Stonebraker : "Implementation of integrity constraints and views by query modification", Proc. ACM-SIGMOD Conf.,San Jose,1975.

Stonebraker76 M. Stonebraker et al. : "The design and implementation of INGRES", Publ., ACM-TODS Vol. 1-3, Sept 1976.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique